

**Arab International University**

**Faculty of Informatics and Communication Engineering**

**Junior Project Report on**

**Vulnerability Detection**

Submitted to

Department of Informatics Engineering

in partial fulfillment of the requirement for theDegree of Bachelor in

**Informatics Engineering**

Submitted by

**Mohamed Mjd alhafi Mohammad Hammade  
Abd Alrahman Damman**

Under the Supervision of

**Eng. Khloud Al Jallad  
  
February 2022**

**© AIU Arab International University**

**All Rights Reserved**

**February 2022**

Faculty of Informatics & Communication Engineering

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Informatics Engineering for acceptance, a project report entitled Project Title in English Submitted by: Mohamed Mjd Alhafi, Mohammad Hammade and Abd Alrahman Damman in partial fulfilment for the degree of Bachelor of Engineering in Informatics.

**Supervisor Head of Department**

**Eng. Khloud Al Jallad Dr. Said Desouki**

**Arab International University**

 The Arab International University (AIU) is a private Syrian university established in 2005. Its academic plans and the documents issued by it are approved and certified by the Ministry of Higher Education in the Syrian Arab Republic.

**The university works to achieve the following goals:**

* Preparing a distinguished generation of university graduates who are able to meet and advance the specific needs of society.
* Contributing to theoretical and applied scientific research that serves the purposes of national development. Work is being done to urge professors and academic staff to scientific research and participate in conferences and seminars that organize research.
* Achieving partnership with prestigious Arab and foreign universities with the aim of continuous development and modernization of academic work and conducting joint scientific research.
* Attracting distinguished academic and research competencies by providing the appropriate environment for their work.

**The Arab International University** is one of the first Syrian universities that have been established and inaugurated. It has been able to attract distinguished educational, research and administrative competencies, to create an integrated edifice from the academic, organizational and administrative aspects. It was able to graduate cadres of distinguished innovators by providing an educational environment based on unique qualitative and material ingredients, including:

* Modern and advanced study plans based on the credit hour system.
* Carefully selected educational cadres.
* Modern scientific laboratories and a laboratory for electronic libraries.
* Physical and moral incentives for students.
* Application of interactive teaching methods.
* Academic and educational guidance and counseling.
* A wide range of scientific cooperation agreements with local, regional and international universities of reputable reputation.
* Multiple agreements and memoranda of understanding with many civil society organizations.
* A proper campus with all the facilities of science, sports and entertainment, which we encourage you to visit and learn about their features.
* Student activities and clubs of all kinds: athletic, cultural, scientific and social.

**The Arab International University** life years are a time to invest in a student's future. The knowledge and experience that students acquire in the lecture hall and laboratories will help them in developing themselves. They will provide them with reasons for success in the chosen specialty. The student activities will help in expanding students’ horizon. The activities of training, clubs and sports will enable students to develop their talents, and may even help in discovering new talents.

Students could invest time, mind and spirit in our university in order to reap the benefits of works and the time devoted in the coming years. We will be by our students in every step of their way.

*الحمدالله الذي وفقنا في إنهاء هذا البحث العملي، والذي أنعم علينا ينعمة الصبر والعافية والصحة.*

*وأتقدم بالشكر والتقدير إلى المهندسة الرائعة خلود الجلاد التي أعطتنا جميع النصائح والإرشادات حتى يكون البحث بصورته النهائية الكاملة.*

*وأيضاً أشكر والدتي خلود الأيوبي ووالدي موفق الحافي اللذان قدما الكثير والكثير من أجل أن أصل إلى هذه المرحلة.*

*وإلى أخي أحمد وأختي دانيا على تشجيعهم المستمر لي.*

*وأيضاً إلى الزميل أحمد الحريري لما قدّمه من عطاء ونصح وتوجيه.*

*محمد مجد الحافي*

**Acknowledgements**

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Eng. **Khloud Al Jallad** for her valuable guidance, encouragement and great dedication for completing this work, her useful suggestions for this whole work and cooperative behavior are sincerely acknowledged.

At the end we would like to express our sincere thanks to all our families, friends and others who helped us directly or indirectly during this project work.

**Abstract**

Application security is an essential part of developing modern software. As the internet increases in complexity, attackers are turning more and more to known security flaws and vulnerabilities in programs themselves. To avoid data breaches, companies need to build security into all the phases of building, testing, and deploying their software. There are different techniques to detect vulnerability such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) but these solutions suffer from high false-positive and high false-negative rates. Researchers have been interested to develop an AI-based system to detect vulnerabilities using Deep Learning models such as Bert, BLSTM, etc. In this project, two deep learning models were developed for vulnerability detection in C/C++ source codes, one is to detect if source code contains any vulnerability (binary classification model) and the other (multiclass classification model) to classify this vulnerability. The binary classification model consists of CNN and the multiclass classification model consists of Convolutional+ LSTM. Both of them were trained on SySeVR dataset [1]. Results over the dataset, for the binary classification model, the accuracy is 99%. For multiclass classification model, the accuracy is 98% to classify over 50 different types of vulnerabilities.

**Contents**

[Introduction 2](#_Toc95457421)

[Chapter 1: Project Description 4](#_Toc95457422)

[1.1 Background 4](#_Toc95457423)

[1.2 Problem Statement 4](#_Toc95457424)

[1.3 Project Objective 4](#_Toc95457425)

[1.4 Project Scope 5](#_Toc95457426)

[1.5 Project Features 5](#_Toc95457427)

[1.6 Project Feasibility 5](#_Toc95457428)

[1.7 System Requirements 5](#_Toc95457429)

[Chapter 2: Theoretical Study 6](#_Toc95457430)

[2.1 Introduction 6](#_Toc95457431)

[2.2 Important of Vulnerabilities Detection 6](#_Toc95457432)

[2.3 Vulnerabilities Examples 6](#_Toc95457433)

[2.4 Traditional Vulnerabilities Detection Techniques 8](#_Toc95457434)

[2.5 Detect Vulnerability using Deep Learning 9](#_Toc95457435)

[2.6 Similar Applications 10](#_Toc95457436)

[Chapter 3: Literature Review 12](#_Toc95457437)

[3.1 Related Works 12](#_Toc95457438)

[3.1.1 Dataset 12](#_Toc95457439)

[3.1.2 Previous work 14](#_Toc95457440)

[3.2 Comparison 20](#_Toc95457441)

[Chapter 4: System Analysis 21](#_Toc95457442)

[4.1 Functional Requirements 21](#_Toc95457443)

[4.2 Non-Functional Requirements: 21](#_Toc95457444)

[4.3 Use cases 21](#_Toc95457445)

[4.4 Context Diagram 22](#_Toc95457446)

[Chapter 5: System Design 23](#_Toc95457447)

[5.1 Block Diagram 23](#_Toc95457448)

[5.2 Data Flow Diagram 23](#_Toc95457449)

[5.3 Collaboration Diagram 24](#_Toc95457450)

[5.4 Sequence Diagram 26](#_Toc95457451)

[5.5 Activity Diagram 28](#_Toc95457452)

[5.6 Web Pages Design 30](#_Toc95457453)

[5.6.1 Home 30](#_Toc95457454)

[5.6.2 Github Repository 31](#_Toc95457455)

[5.6.3 Code Editor & Results 32](#_Toc95457456)

[5.6.4 About as 33](#_Toc95457457)

[5.7 Dataset 34](#_Toc95457458)

[5.8 Models 39](#_Toc95457459)

[5.8.1 Binary classification model 39](#_Toc95457460)

[5.8.2 Multiclass classification model 42](#_Toc95457461)

[Chapter 6: Result & Conclusion 44](#_Toc95457462)

[6.1 Results 44](#_Toc95457463)

[6.1.1 Binary Classification Model 44](#_Toc95457464)

[6.1.2 Multiclass Classification Model 46](#_Toc95457465)

[6.2 Results Comparison 48](#_Toc95457466)

[6.3 Code snip 49](#_Toc95457467)

[6.3.1 binary classification model 49](#_Toc95457468)

[6.3.2 multiclass classification model 50](#_Toc95457469)

[Conclusion and Future Works 51](#_Toc95457470)

[References 52](#_Toc95457471)

**Abbreviations**

AIU Arab International University

RT Report Template

SVM Support Vector Machine

RF Random Forest

BERT Bidirectional Encoder Representations from Transformers

GRU Gated recurrent unit

NLP Natural Language Processing

CWE Common Weakness Enumeration

LSTM Long Short Term Memory

ReLU Rectifier Linear Unit

CVE Common Vulnerability and Exposure

SARD Software Assurance Reference Dataset

NVD National Vulnerability Database

DT Decision Tree

LR Logistic Regression

CNN Convolutional Neural Network

SMOTE Synthetic Minority Oversampling Technique

**Keywords**

Security, Vulnerability, Attacks, Deep learning, Natural Language Processing.

# Introduction

Attacks are defined as the exploitation of vulnerabilities in a system, where these vulnerabilities are caused by human errors with no malicious intent. The attacker relies heavily on these vulnerabilities when performing the hack in order to damage the system. In conclusion, it figures out that vulnerabilities are the main gateway for the occurrence of combustion. With the considerable development in technology in the last decades, electronic attacks pose a real danger to systems, the most important of which is banks. Therefore, it was necessary to have tools that help developers detect vulnerabilities in their source codes, in order to make their system more secure and not contain any gateway that allows attackers to enter the system.

# Chapter 1: Project Description

## Background

Automatic detection of vulnerabilities is a fundamental problem in systems security. As the previous solutions were based on special techniques in the detection suffer from high false-positive/false-negative rates. Therefore, many researchers are moving to take advantage of the great development in deep learning to detect security vulnerabilities, especially with the availability of hardware (GPU) that allows deep learning models to have more robustness.

## Problem Statement

New vulnerabilities will always appear, and hackers will always look for new exploits. Whereas, a vulnerability was discovered in the log4j library, which is written in Java and is widely used in many cloud storage platforms such as Apple iCloud and Steam. This vulnerability allows the hacker to take full control of the server. The vulnerability has existed in the library since 2013, and no one discovered it until November 2021, and this if anything indicates that the vulnerabilities exist in all products and the software has not been spared. Therefore, there must be a tool that guarantees organizations the security of their applications to protect them from intrusions.

## 1.3 Project Objective

The aim of this project is to develop a tool that detects security vulnerabilities in software programs using artificial intelligence concepts. As one of the most common types of attack comes from software vulnerabilities and because of the increasing number of software developed every day. Automatically detecting vulnerabilities in software is a very important research problem that draws extensive attention in recent years due to the substantial losses caused by hacker attacks.

## 1.4 Project Scope

Scan the source code to first detect if the code contains a vulnerability and then classify the vulnerability into one of 50 classes, for example, Stack-based Buffer Overflow, Unchecked Return Value to NULL Pointer Dereference and etc. Provide different ways for submitting the source code: the user can open the repository of his Github account by Visual Studio Code and then choose the function he wants, or he can write the function in code editor in the website.

## 1.5 Project Features

A website that allows users to display their Github repository, scan their own source code that is written in C/C++ and give them results if the code is vulnerable or not, classify the vulnerability if any and return details of the classified vulnerability, visualize their function as a graph to help them to fix their vulnerable code, and developing RESTful API server to send and receive information from deep learning models and the website.

## 1.6 Project Feasibility

This project can be implemented to any source code in C/C++ to predict the class of vulnerability if exist in a short period of time and provide details to help users to well-understand their code and fix it.

## 1.7 System Requirements

Experience with neural networks, natural language processing, and data preprocessing. Using a project management framework such as Scrum which has an iterative and incremental pattern to increase the quality and efficiency of the project. And also, a large dataset for training AI models

# Chapter 2: Theoretical Study

## 2.1 Introduction

Software development can be complex. Errors, faults, and failures are introduced in many stages of the software life cycle. The consequences of a class of system failures, commonly known as software vulnerabilities. They can cause the loss of information, and reduce the value or usefulness of the system [2].

## 2.2 Important of Vulnerabilities Detection

Vulnerabilities are the gate for the hackers and attackers that leads to corporate network data breaches, destruction of cyber-physical systems, or physical harm to people [3]. For example, Hackers exploit the vulnerabilities of web servers and inject malicious code in order to bypass login and gain unauthorized access to backend databases [4], They exploit the vulnerabilities by writing code to target a specific security weakness. They package it into malware called a zero-day exploit. The malicious software takes advantage of a vulnerability to compromise a computer system or cause an unintended behavior [5].

## 2.3 Vulnerabilities Examples

* Buffer overflow [6]: is the probably the best-known form of software security vulnerability. Most software developers know what a buffer overflow vulnerability is, but buffer overflow attacks against both legacy and newly-developed applications are still quite common. Part of the problem is due to the wide variety of ways buffer overflows can occur, and part is due to the error-prone techniques often used to prevent them.
* Cross-site scripting [7] (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.
* SQL injection [8]: is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.
* Format string bugs [9]: is a bug where user input is passed as the format argument to printf, scanf, or another function in that family. The format argument has many different specifies which could allow an attacker to leak data if they control the format argument to printf. Since printf and similar are variadic functions, they will continue popping data off of the stack according to the format.
* Integer overflows [10]: also known as wraparound, occurs when an arithmetic operation outputs a numeric value that falls outside allocated memory space or overflows the range of the given value of the integer. Mostly in all programming languages, integers values are allocated limited bits of storage.

## Traditional Vulnerabilities Detection Techniques

Techniques to detect vulnerabilities are based on Static application security testing (SAST) and dynamic application security testing (DAST) [11]

|  |  |
| --- | --- |
| SAST | DAST |
| White box security testing  The tester has access to the underlying framework, design, and implementation. The application is tested from the inside out. This type of testing represents the developer approach. | Black box security testing  The tester has no knowledge of the technologies or frameworks that the application is built on. The application is tested from the outside in. This type of testing represents the hacker approach. |
| Requires source code  SAST doesn’t require a deployed application. It analyzes the sources code or binary without executing the application. | Requires a running application  DAST doesn’t require source code or binaries. It analyzes by executing the application. |
| Finds vulnerabilities earlier in the SDLC  The scan can be executed as soon as code is deemed feature-complete. | Finds vulnerabilities toward the end of the SDLC  Vulnerabilities can be discovered after the development cycle is complete. |
| Less expensive to fix vulnerabilities | More expensive to fix vulnerabilities |
| Since vulnerabilities are found earlier in the SDLC, it’s easier and faster to remediate them. | Since vulnerabilities are found toward the end of the SDLC, remediation often gets pushed into the next cycle. |
| Typically supports all kinds of software  Examples include web applications, web services, and thick clients. | Typically scans only apps like web applications and web services  DAST is not useful for other types of software. |
| typically result in high false positives, i.e., detect non-vulnerable cases as vulnerable [12]. | suffers from high false negatives, i.e., cannot detect many real vulnerabilities [13]. |

## 2.5 Detect Vulnerability using Deep Learning

Recent progress in Deep Learning (DL), especially in domains like computer vision and natural language processing, has sparked interest in using DL to detect security vulnerabilities automatically with high accuracy. According to Google scholar, 92 papers appeared in popular security and software engineering venues between 2019 and 2020 that apply learning techniques to detect different types of bugs. And it is natural to ask why they are performing so well, what kind of features these models are learning, and most importantly, whether they can be used effectively and reliably in detecting real-world vulnerabilities. For instance, the generalizability of a DL model is limited by implicit biases in the dataset, which are often introduced during the dataset generation/curation/labeling process and therefore affect both the testing and training data equally (assuming that they are drawn from the same dataset). These biases tend to allow DL models to achieve high accuracy in the test data by learning highly idiosyncratic features specific to that dataset instead of generalizable features [13].

## 2.6 Similar Applications

There are a lot of applications that help the developers to easily detect the vulnerabilities. Some of these applications are based either on static analysis or dynamic analysis and the others are based on AI.

* Fortify Static Code Analyzer[[1]](#footnote-1): produce understandable and traceable vulnerability information and make it easy to clean out false positives manually [14].
* Hakiri[[2]](#footnote-2): is a commercial tool that offers dependency checking for Ruby and Rails-based GitHub projects using static code analysis. It uses NVD and the Ruby Advisory Database. Hakiri statically analyzes every GitHub commit and pull request for 32 different types of vulnerabilities such as SQL injection, XSS [15].
* WhiteSource3: enforces policies automatically, spotting problems before they surface or remediating as soon as they are detected. It then secures the developers from vulnerabilities and enforces license policies throughout the software development lifecycle [16].
* Netsparker[[3]](#footnote-3)[[4]](#footnote-4): is a dynamic analysis security testing application security solution that can help to find certain vulnerabilities in web applications, upon identifying a vulnerability, the scanner generates a proof of exploit that confirms it is not a false positive, improving automation and scalability [17].
* ShiftLeft[[5]](#footnote-5): ShiftLeft CORE provides accurate results and contextual education to developers so they quickly reduce security risk without losing focus on delivery. The CORE workflow inserts into pull requests with fast feedback so developers can find and fix vulnerabilities within the code they are already working on [18].
* YAGAAN[[6]](#footnote-6): The Yag-Suite is the solution for application security that brings new capabilities to source code analysis. Its aim is, through a decision-making approach, to support developers and reviewers in their efficient targeting of the source code vulnerabilities which are the most relevant to fix [19].
* PT Application Inspector[[7]](#footnote-7): is a source code analyzer. A unique combination of scanning methods—static application security testing (SAST), dynamic application security testing (DAST), interactive application security testing (IAST), software composition analysis (SCA) [20].
* Snyk Code[[8]](#footnote-8): follows a fundamentally new concept for static code analysis. Snyk Code is part of the Snyk platform, helping developers build software securely and find more vulnerabilities during real-time scanning code [21].

# Chapter 3: Literature Review

## 3.1 Related Works

### 3.1.1 Dataset

Many datasets are created in this domain, this chapter includes a description and comparison of the most recent datasets used in the researches.

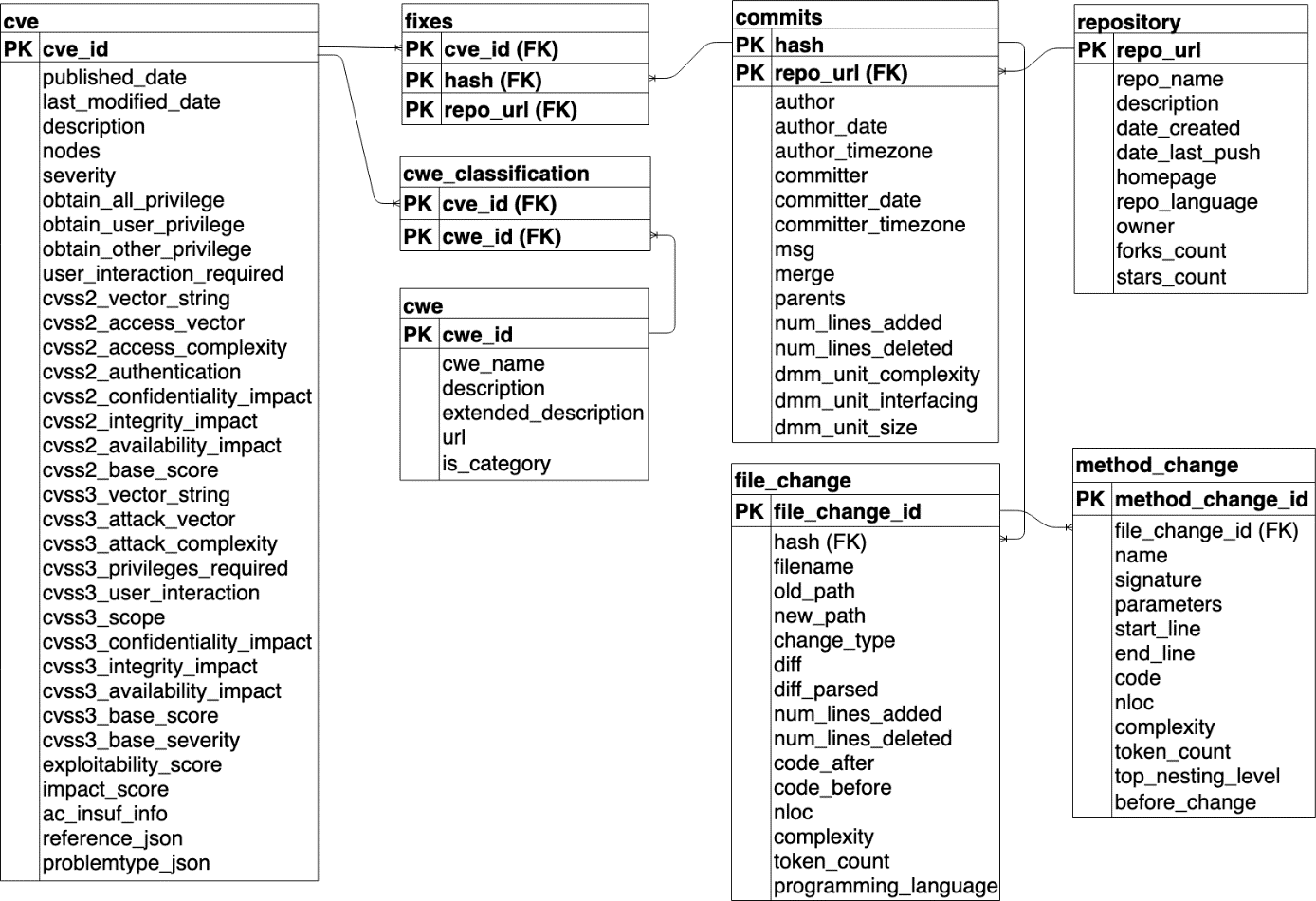
* The Software Assurance Reference Dataset (SARD) [22] is a growing collection of over 170 000 programs with precisely located bugs. The programs are in C, C++, Java1, PHP, and C# and cover more than 150 classes of weaknesses, such as SQL injection, cross-site scripting (XSS), buffer overflow, and use of a broken cryptographic algorithm.
* The National Vulnerability Database (NVD) [23] is a public data source that maintains standardized information about reported software vulnerabilities. Since its inception in 1997, NVD has published information about more than 43,000 software vulnerabilities affecting more than 17,000 software applications.
* CVEFixes [24] is a comprehensive vulnerability dataset that is automatically collected and curated from Common Vulnerabilities and Exposures (CVE) records in the public [U.S. National Vulnerability Database (NVD)]. The goal is to support data-driven security research based on source code and source code metrics related to fixes for CVEs in the NVD by providing detailed information at different interlinked levels of abstraction, such as the commit-, file-, and method level, as well as the repository- and CVE level. CVEFixes dataset is structured as a relational database consisting of multiple tables where each table presents artifacts at each specific abstraction level. figure 1 shows how the tables are organized and connected. 

Figure 1 CVEfixes dataset structure

* The ManySStuBs4J [25] corpus is a collection of simple fixes to Java bugs, designed for evaluating program repair techniques. Data includes a collection of all bug-fixing changes using the SZZ heuristic, and then filter these to obtain a data set of small bug-fix changes. These are single statement fixes, classified where possible into one of 16 syntactic templates which call SStuBs. The dataset contains simple statement bugs mined from open-source Java projects hosted in GitHub. There are two variants of the dataset. One mined from the 100 Java Maven Projects and one mined from the top 1000 Java Projects.
* The ESC [26] (Ethereum Smart Contracts) dataset consists of 40,932 smart contracts from Ethereum with roughly 307,396 functions in total. Among the functions, around 5,013 functions possess at least one invocation to call. Value, making them potentially affected by the reentrancy vulnerability. Around 4,833 functions contain the block. timestamp statement, making them susceptible to the timestamp dependence vulnerability. Ethereum is a decentralized blockchain platform that can build a broad scope of applications.
* The VSC [27] (VNT Chain Smart Contracts) dataset contains all the available 4,170 smart contracts collected from the VNT Chain network, which overall contain 13,761 functions. VNT Chain is an experimental public blockchain platform proposed by companies and universities from Singapore, China, and Australia.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Size | Year | Samples number | Programming  Language | Label type |
| CVEFixes [24] | 1.1GB | 2021 | 61K | C/C++ | Binary |
| ReVeal  FFMPeg+Qemu [28] | N/A | 2020 | 40K | C | Binary |
| ManySStuBs4J [25] | 887MB | 2019 | 63K | Java | Multiple class |
| Drapper VDISC [29] | 1GB | 2018 | 1.27M | C/C++ | Multiple class |
| NVD [30] | N/A | 1999 | 100K | C/C++ | Multiple class |
| ESC [26] | 113MB | N/A | 307K | C/C++ | Binary |

### 3.1.2 Previous work

#### **3.1.2.1 Graph based Solutions**

A lot of successful works on vulnerability classification based on graph because the code can be represented as Abstract Syntax Tree (AST) which is a type of graph. The structure of the graph can capture the decencies and relation among nodes a lot of researches use graph as in [13] [31] [32].

##### **3.1.2.1.1 Machine learning models**

Machine learning models are used to classify vulnerabilities from codes in some researches such as random forest and SVM [13] in while logistic regression was used in [31]. The feature engineering stage extracted from code includes the following features:

1. Gated Graph Neural Networks (GGNN)

Saikat Chakraborty et al. [13] in used Code Property Graph to represent the original code, then they use GGNN that assigns a Gated Recurring Unit (GRU) to update the current vertex embedding by assimilating the embedding of all its neighbors.

##### **3.1.2.1.2 Deep learning models**

Some researchers used deep learning-based vulnerability detection to relieve human experts from the tedious and subjective task of manually defining features such as MLP in [13] and CNN, DBN, LSTM, GRU in [31] and in [32] encoder-decoder architecture was used while RNN, LSTM, GRU, GCN, DR-GCN as in [33].

1. Abstract Syntax Tree (AST)

Zhen Lis et al. in [31] represented program as vector using the notation of Syntax-based Vulnerability Candidates (SyVCs) and Semantics-based Vulnerability Candidates (SeVCs) depending on Abstract Syntax Tree (AST). Anshul Tanwa et al. in [32] used Abstract Syntax Tree (AST) to extract features from the code then embedded the features vector to fit them in encoder that consists of self-attention module, BLSTM and Conv layer.

2. Expert Pattern Extraction

Design new patterns for the vulnerabilities, and implement them to automatically extract these patterns as in [33].

3. Contract Graph Construction and Normalization

A contract graph construction with normalization has been used to extracts the control flow and data flow semantics from the source code and highlights the critical nodes as in [33].

4. Graph + Expert patterns

CGE (Combining Graph feature and Expert patterns) as in [33].

#### **3.1.2.2 NLP based solutions**

Several works have been done using natural language processing as in [34] [35] [36].

##### **3.1.2.2.1 Machine learning models**

Machine learning models are used to classify vulnerabilities from codes in some researches such as random forest in [35]. The feature engineering stage extracted from code includes the following features:

1. Bag of Words.

Rebecca L. Russell et al. in [35] proposed using (BOW) because it is simple representing text data and has seen great success in problems such as language modeling and document classification.

##### **3.1.2.2.2 Deep learning models**

Some researchers used deep learning-based vulnerability detection to relieve human experts from the tedious and subjective task of manually defining features such as LSTM in [36] and BLSTM as in [14]. The feature engineering stage for extracted from code include the following features:

1. BERT

Noah Ziems and Shaoen Wu. In [36] used BERT because it was designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context.

2. Tokenization

Noah Ziems and Shaoen Wu. In [36] also tokenized the input source code into words called tokens. These tokens help in understanding the context or developing the model for the NLP. Rebecca L. Russell et al. in [35] also applied Convolution Neural Network (CNN) on tokens because it automatically detects the important features without any human supervision and they also used Recurrent Neural Network (RNN) because its architecture allows to exhibit temporal behavior and capture sequential data which makes it a more 'natural' approach when dealing with textual data since text is naturally sequential.

3. Code Gadget:

In order to represent programs in vectors that are suitable for the input to neural networks, in [34] They proposed transforming programs into a representation of code gadget, which is composed of a number of program statements (i.e., lines of code), which are semantically related to each other in terms of data dependency or control dependency.

4. Key Point:

In order to generate code gadgets, in [34] they proposed the heuristic concept of key point, which can be seen as a “lens” through which programs can be represented from a certain perspective. For example, for vulnerabilities that are caused by improper uses of arrays, the key points are the arrays.

##### **3.1.2.2.3 Deep Learning & Machine Learning models.**

Some researchers used deep learning such as CNN and RNN and machine learning as Random Forest as in [35]. The feature engineering stage for extracted from code include the following features:

1. Embedding: The tokens making up the lexed functions are first embedded into a fixed k-dimensional representation as in [35].

2. Pooling: As the length of functions found in the wild can vary dramatically, both the convolutional and recurrent features are maxpooled along the sequence length in order to generate a fixed-size representation as in [35].

In they found that using the neural features (outputs from the sequence-maxpooled convolution layer in the CNN and sequence-maxpooled output states in the RNN) as inputs to a powerful ensemble classifier such as random forest or extremely randomized trees yielded the best results on our full dataset.

Comparing the results of the vulnerability detection in previous researches.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Paper | Dataset | Model | Result | | Publishing date |
| Accuracy | F1 |
| Applying CodeBERT for Automated Program Repair of Java Simple Bugs [37] | ManySStuBs4J | Code BERT | 68% | - | 2021 |
| Deep Learning based Vulnerability Detection: Are We There Yet? [13] | ReVeal | RF | 85% | 25% | 2020 |
| MLP | 84% | 36% |
| SVM | 82% | 39% |
| ReVeal | 84% | 41% |
| FFMpeg + Qemu | RF | 57% | 52% |
| MLP | 61% | 59% |
| SVM | 61% | 60% |
| ReVeal | 62% | 64% |
| SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities [31] | SySeVR | LR | 92 % | 62% | 2021 |
| CNN | 95% | 81% |
| MLP | 93% | 68% |
| DBN | 91% | 63% |
| LSTM | 95% | 79% |
| GRU | 95% | 81% |
| BLSTM | 96% | 83% |
| BGRU | 96% | 83% |
| Security Vulnerability Detection Using Deep Learning Natural Language Processing [36] | NVD/SARD | LSTM | 72% | - | 2021 |
| BLSTM | 79% | - |
| BERT | 85% | - |
| BERT+LSTM | 93% | - |
| BERT+BLSTM | 93% | - |
| LSTM | 72% | - |
| BLSTM | 79% | - |
| Multi-context Attention Fusion Neural Network for Software Vulnerability Identification [32] | SARD | Attention fusion | - | 99% | 2021 |
| A Deep Learning-Based System for Vulnerability Detection [34] | CVEFixes | BLSTM | - | 95% | 2018 |
| Automated Vulnerability Detection in Source Code Using Deep Representation Learning [35] | Draper VDISC | BOW+RF | 89% | 78% | 2018 |
| RNN | 90% | 80% |
| CNN | 94.4% | 84% |
| CNN+RF | 91.6% | 82% |
| RNN+RF | 91.4% | 81% |
| Combining Graph Neural Networks with Expert Knowledge for Smart Constract Vulnerablility Detection [33] | ESC &  VSC | RNN | 49% | 45% | 2021 |
| LSTM | 53% | 54% |
| GRU | 54% | 54% |
| GCN | 77% | 71% |
| CGE | 89% | 87% |

## 3.2 Comparison

Source code is raw text data, and as machine learning models only deal with numeric values. So, there are various techniques to convert source code into numerical features. The most common solution is to tokenize [38] the source code, In other words to replace each token (single word) into a unique ID But there are a lot of disadvantages in this way because the syntactic and semantics dependencies between lines of code are never obtained, which it's so important to distinguish between different vulnerabilities classes. Not only the above issue needs to be solved but also all the source codes must be of the same length. Another solution, using a Bag Of Word [39] which computes the frequency of the words, and then each sentence is represented (source code in our case) in a vector that contains the frequency of each word in our source code. This solution also does not capture dependency between lines Also, it needs too huge amounts of memory because each vector has a length of the number of different words in the whole source codes. Also, the order of words is lost which may play a vital role to detect vulnerabilities. On other hand to maintain full utilization of syntactic and semantics dependencies by representing source code as a graph to use Graph Neural Network (GNN) for getting powerful features either at level of graph or nodes[40]. There are a different ways to extract the graph from the source code such as Abstract Syntax Tree (AST) [41], Control Flow Graph (CFG) [42], and Code Property Graph (CPG) [43]. A lot of GNN models were used such as Graph Convolution Network (GCN) which takes the node's feature vector and its neighbors' features then apply a neural network [44]. Or more complicated GNN architecture such as Graph Attention Network (GAN) which uses an attention mechanism to obtain the final representation of the graph [45]. As can be seen from the previous work results when researchers applied solutions based on graphs, they obtained higher accuracy than other solutions. Also, using a more complex architecture model helping them to improve the accuracy such as the encoder-decoder model which is based on recurrent, convolution, and self-attention network in [32].

# Chapter 4: System Analysis

## 4.1 Functional Requirements

The user should be able to do the following:

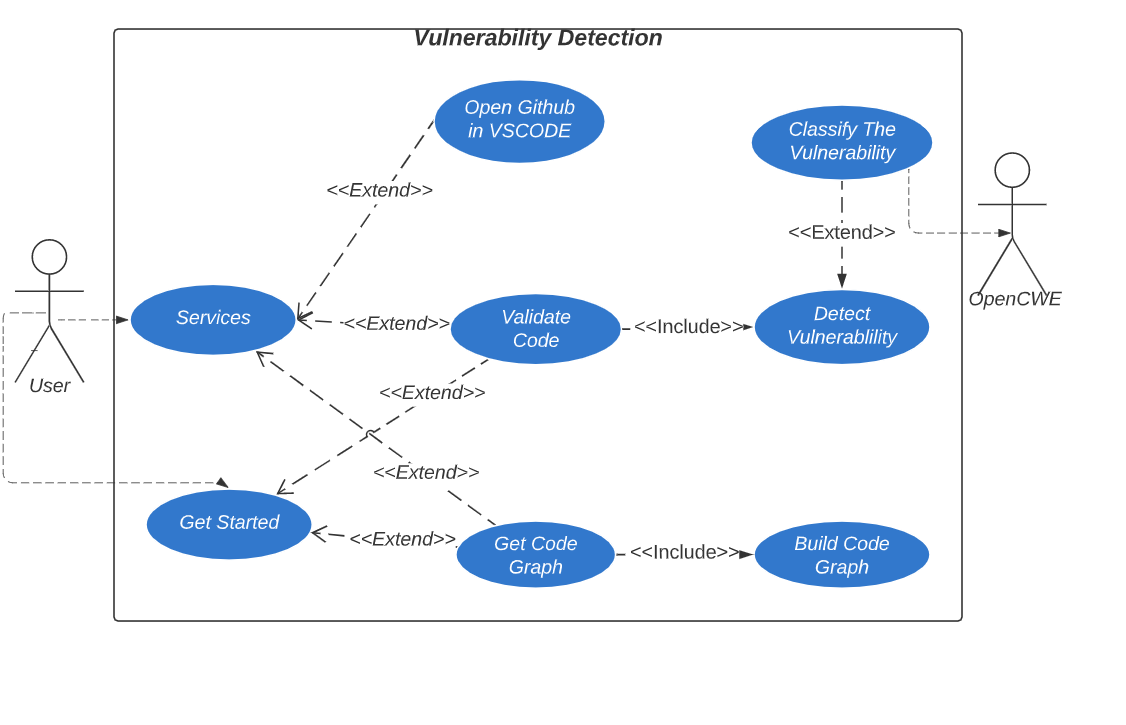
1. Open Github repository by Visual Studio Code.
2. Validate code
3. Draw code graph

## 4.2 Non-Functional Requirements:

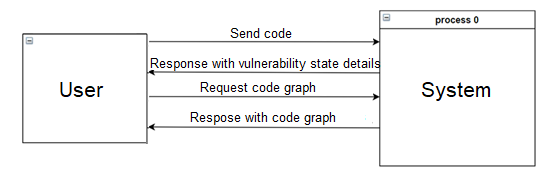
1. Security
2. Performance
3. Transparency
4. Availability
5. Reliability

## 4.3 Use cases

Use case is a list of actions or event steps typically defining the interactions between a role and a system to achieve a goal. The actor can be a human or other external system. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in the Systems Modeling Language or as contractual statements. After specifying the functional requirements of the application, this study represents the use case diagram that connects those functionalities together in order to show how the flow of information within the application is done.

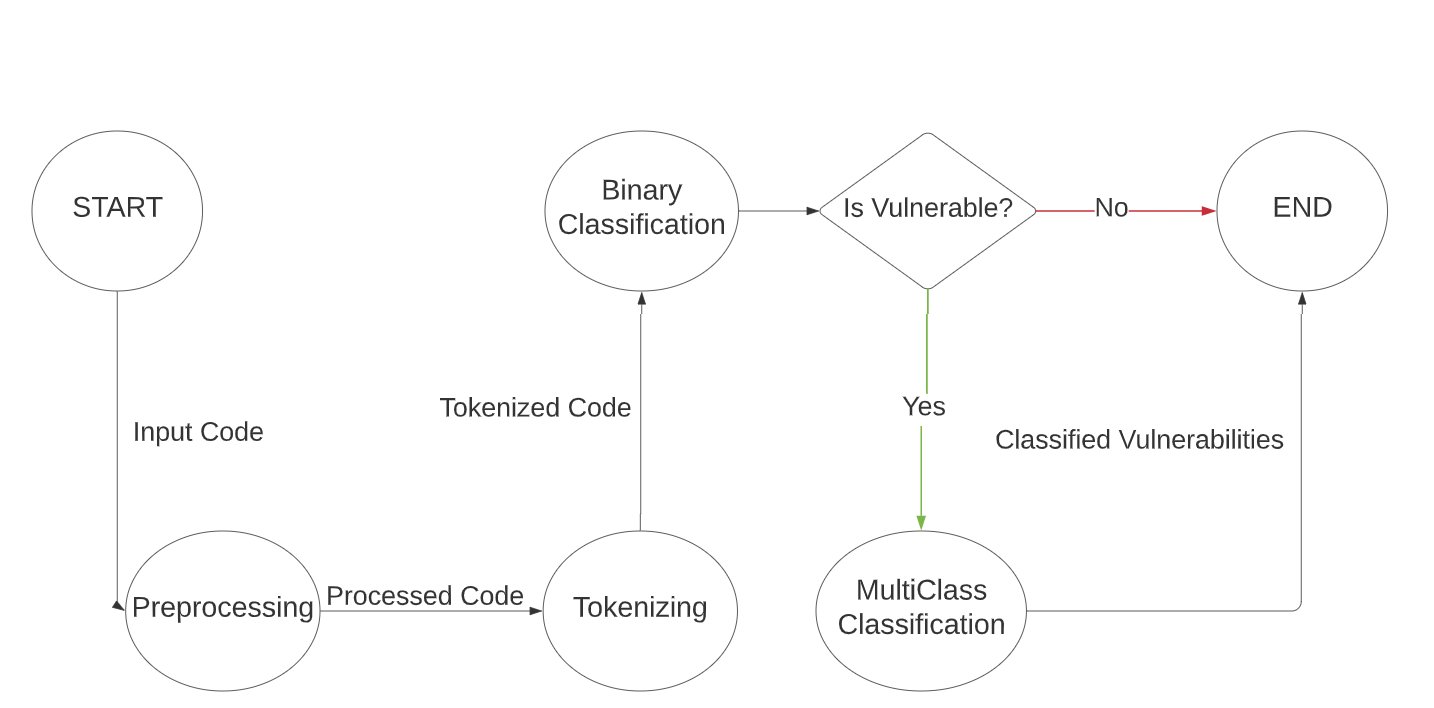


## 4.4 Context Diagram

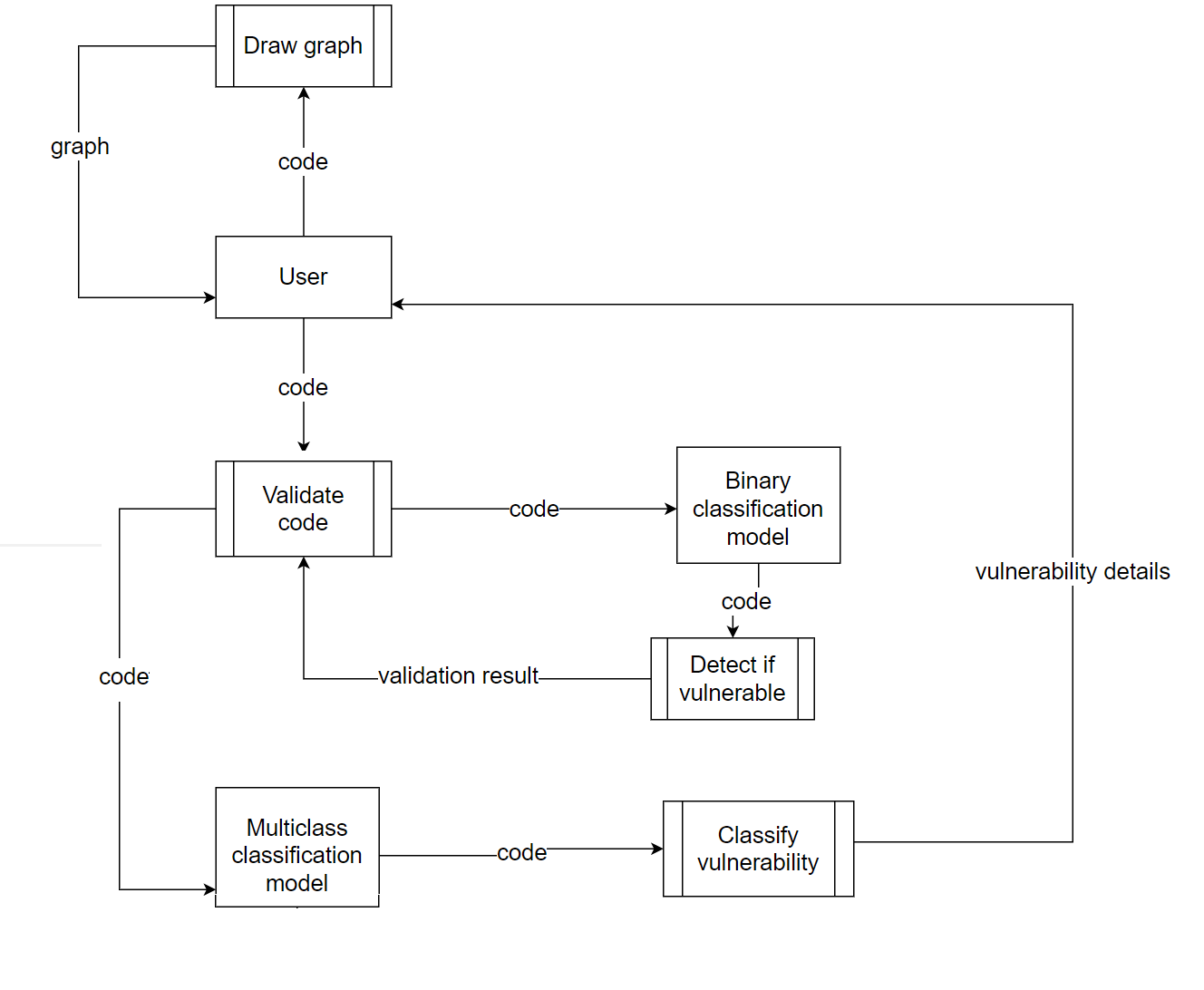


# Chapter 5: System Design

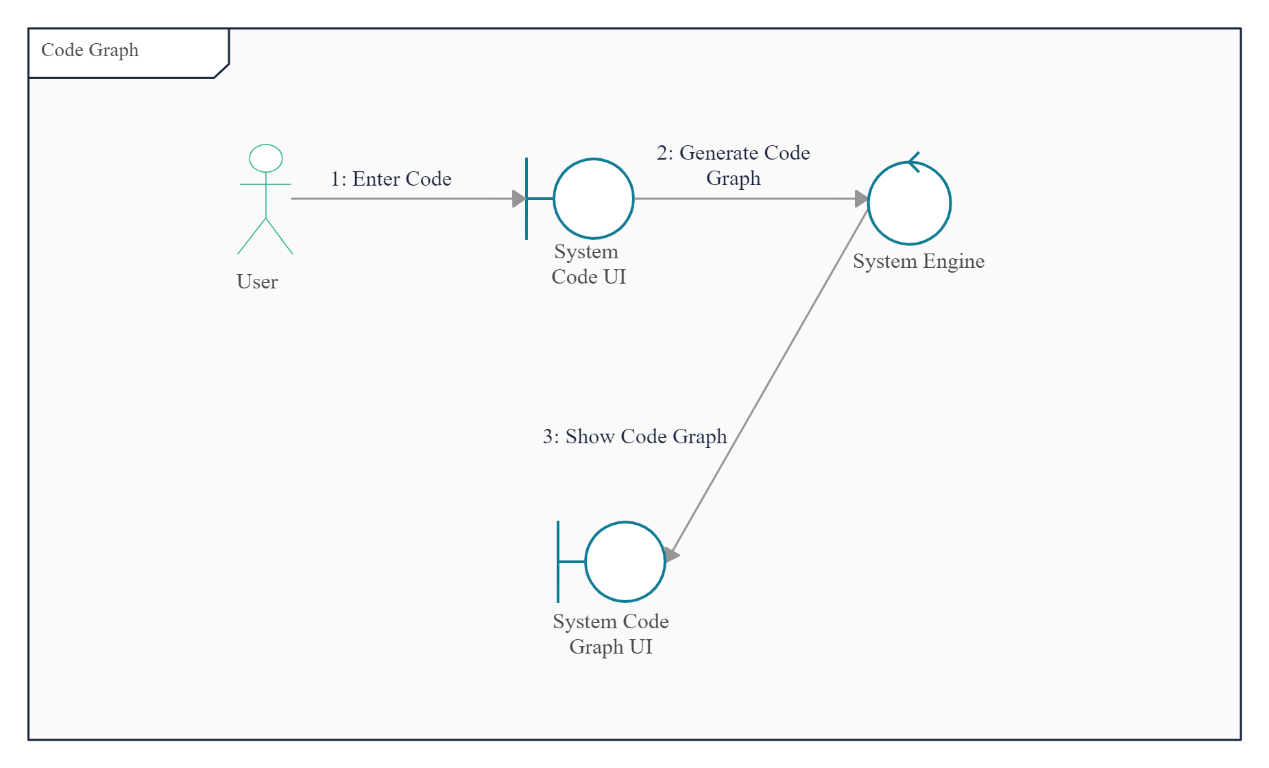
## 5.1 Block Diagram

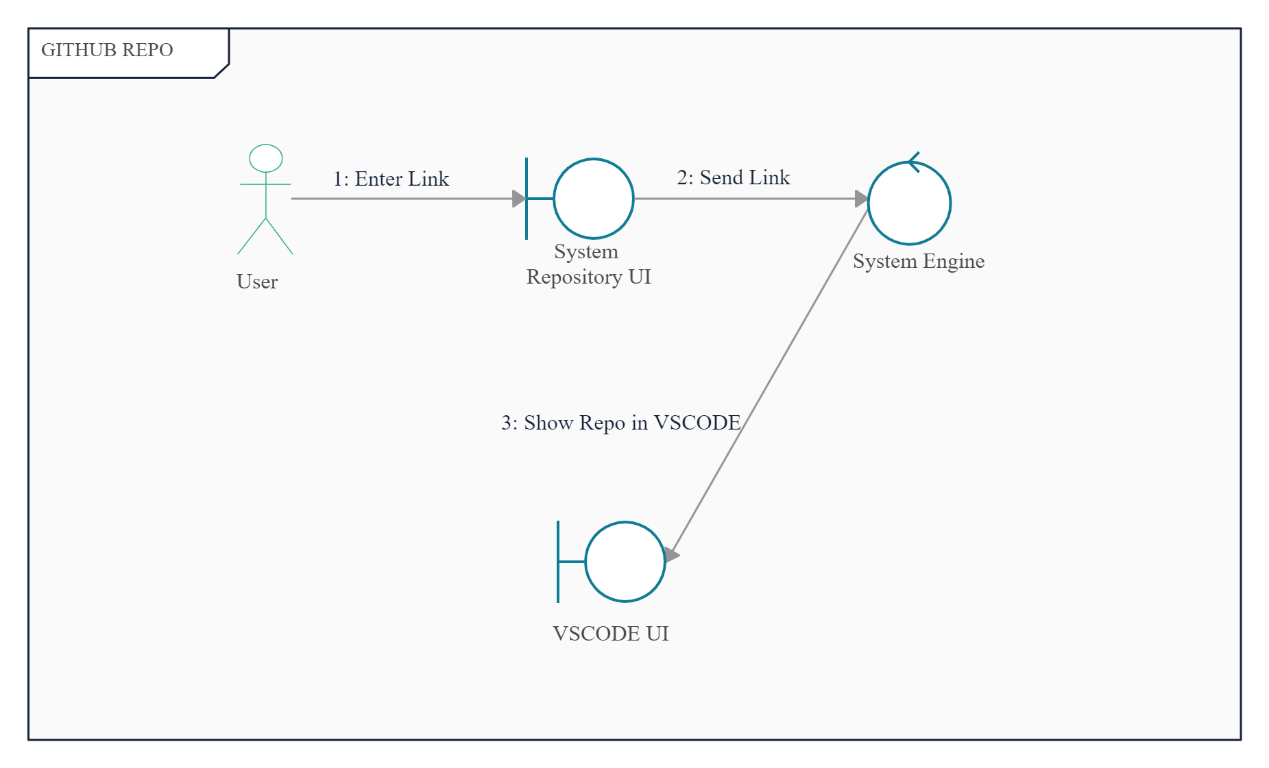


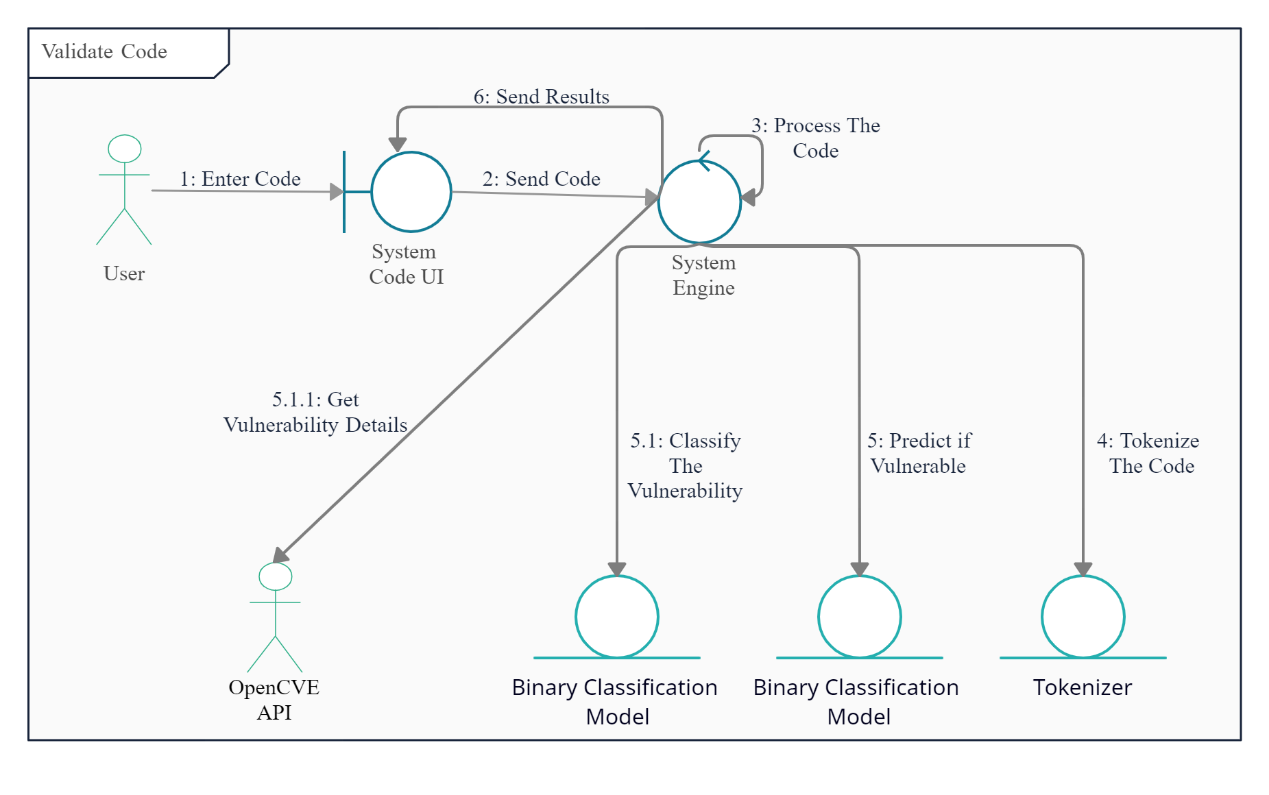
## 5.2 Data Flow Diagram



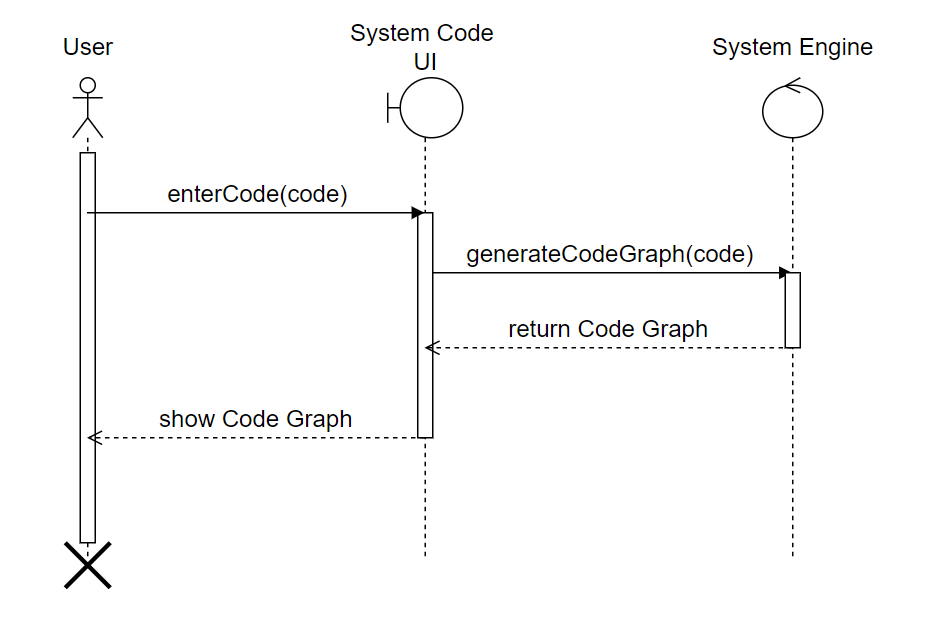
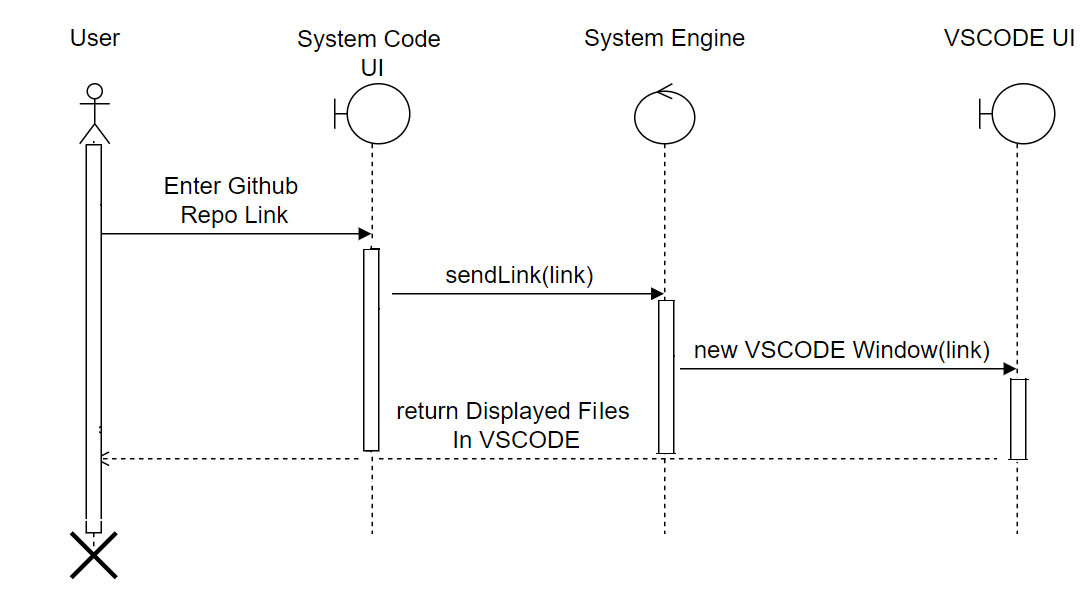
## 5.3 Collaboration Diagram

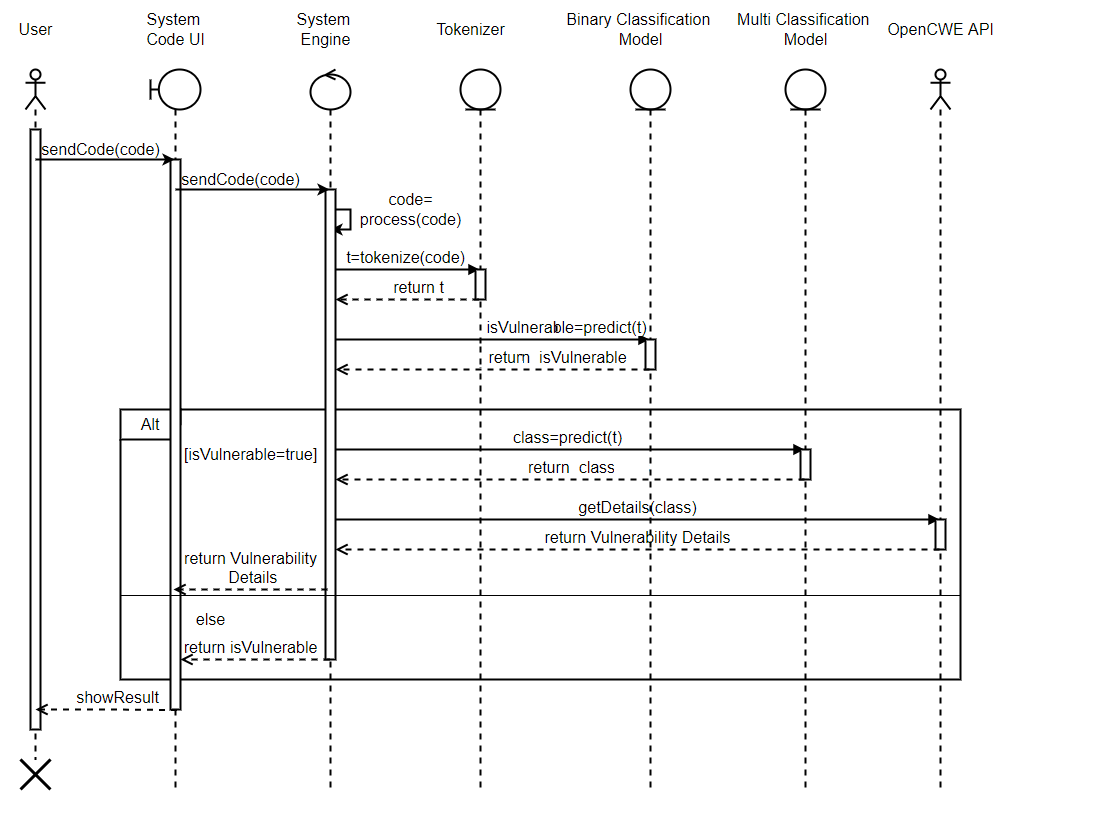




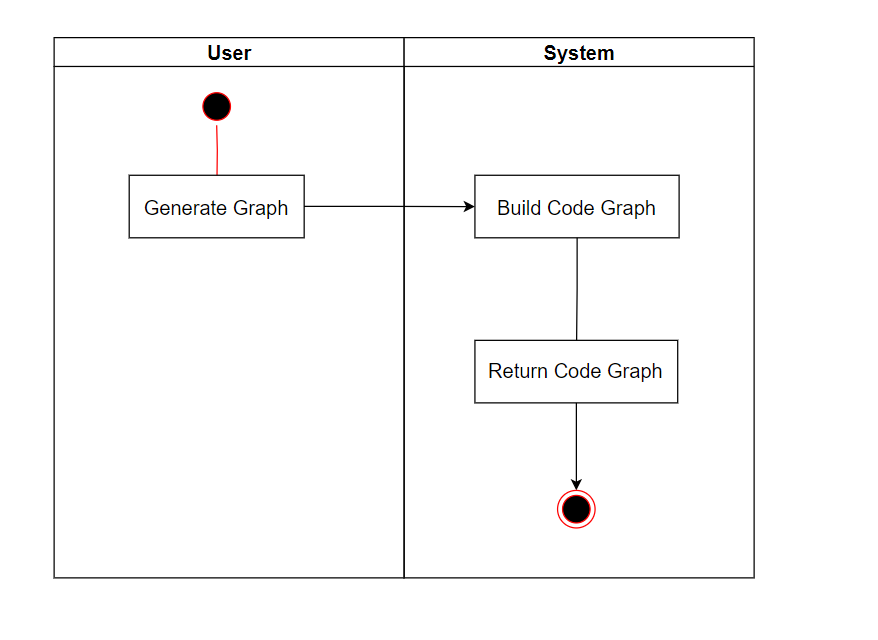


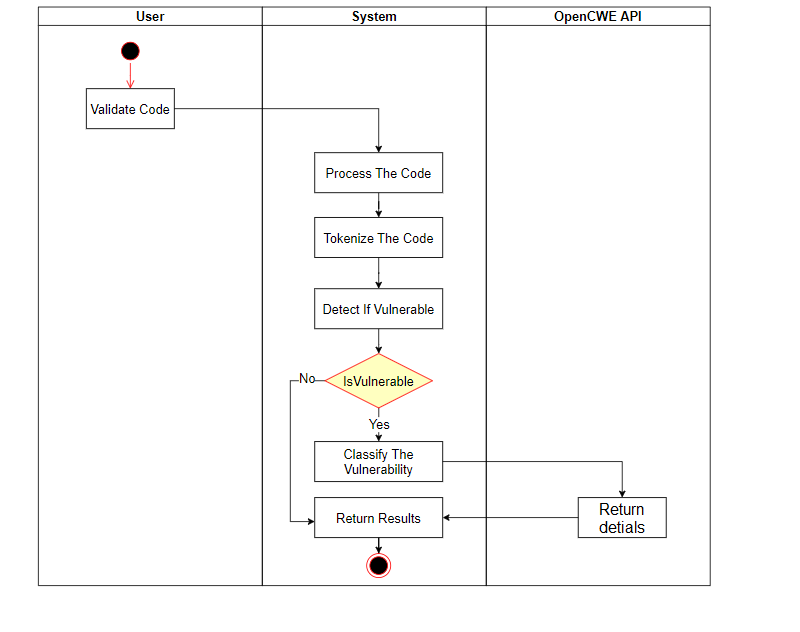
## 5.4 Sequence Diagram





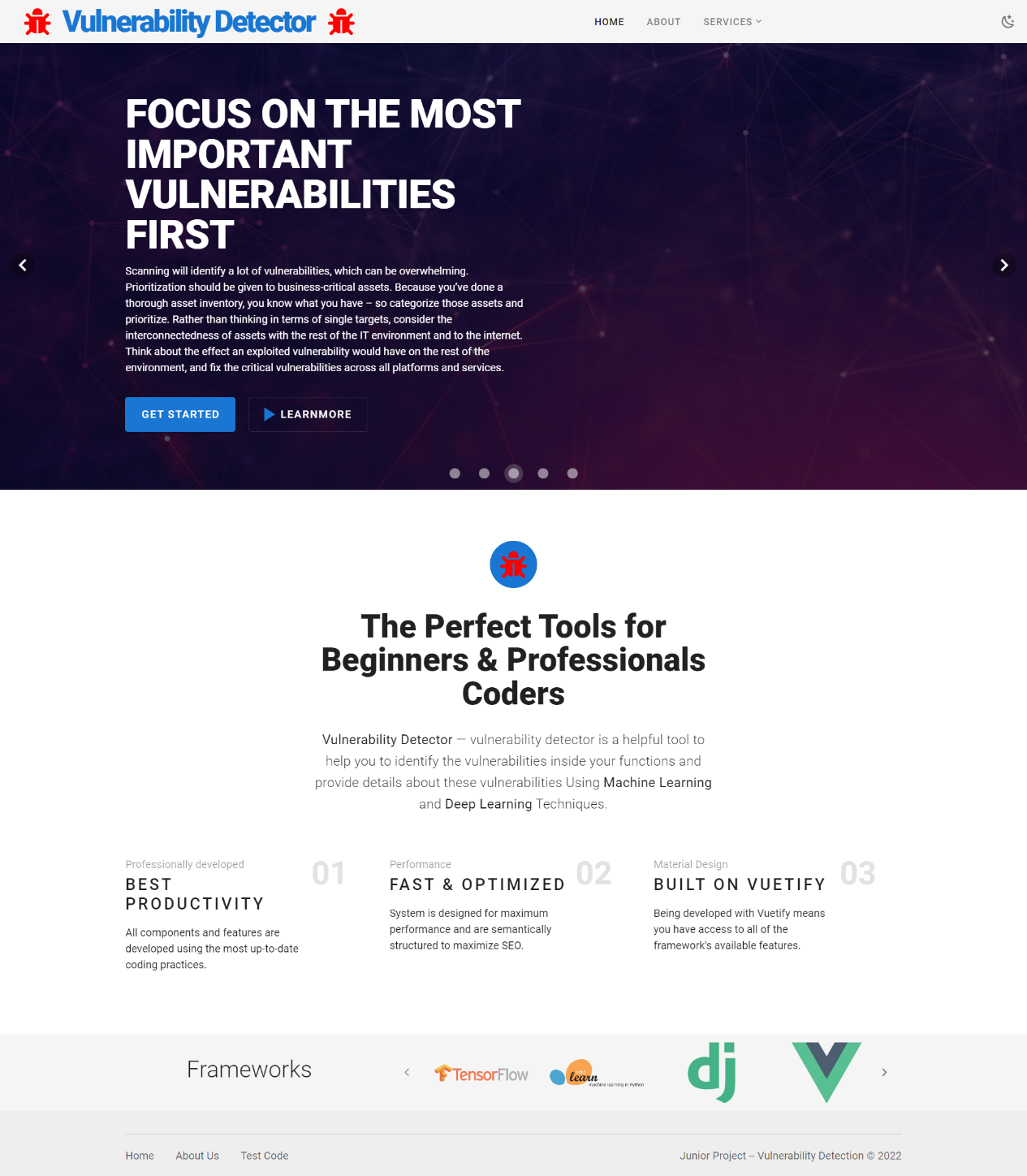
## 5.5 Activity Diagram



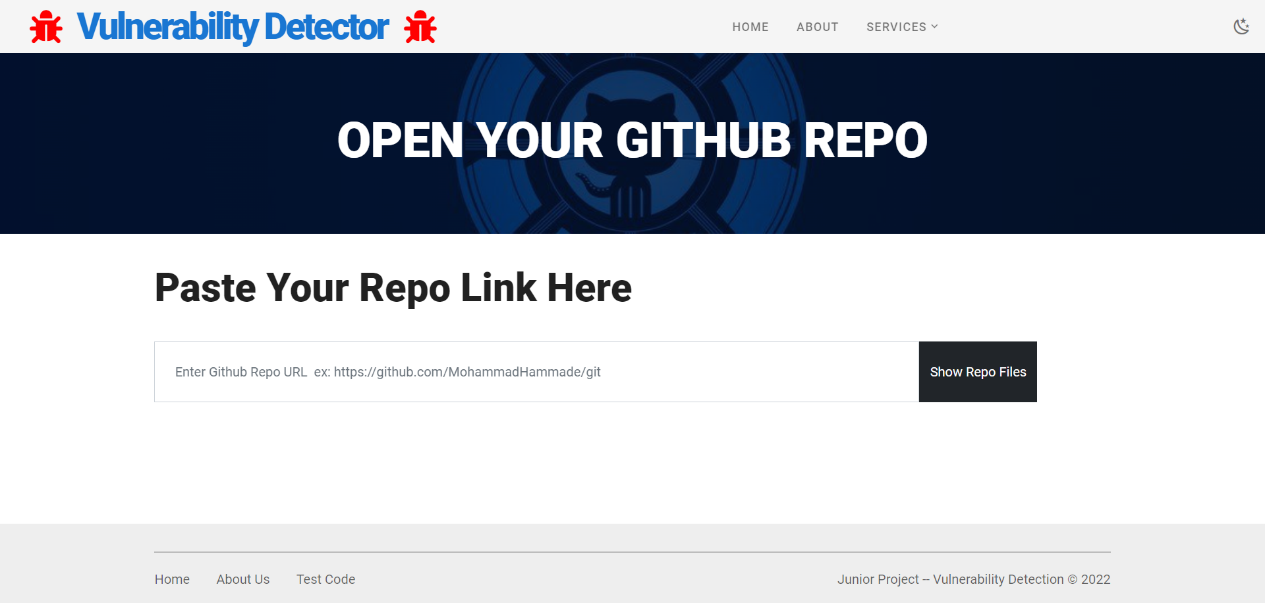


## 5.6 Web Pages Design

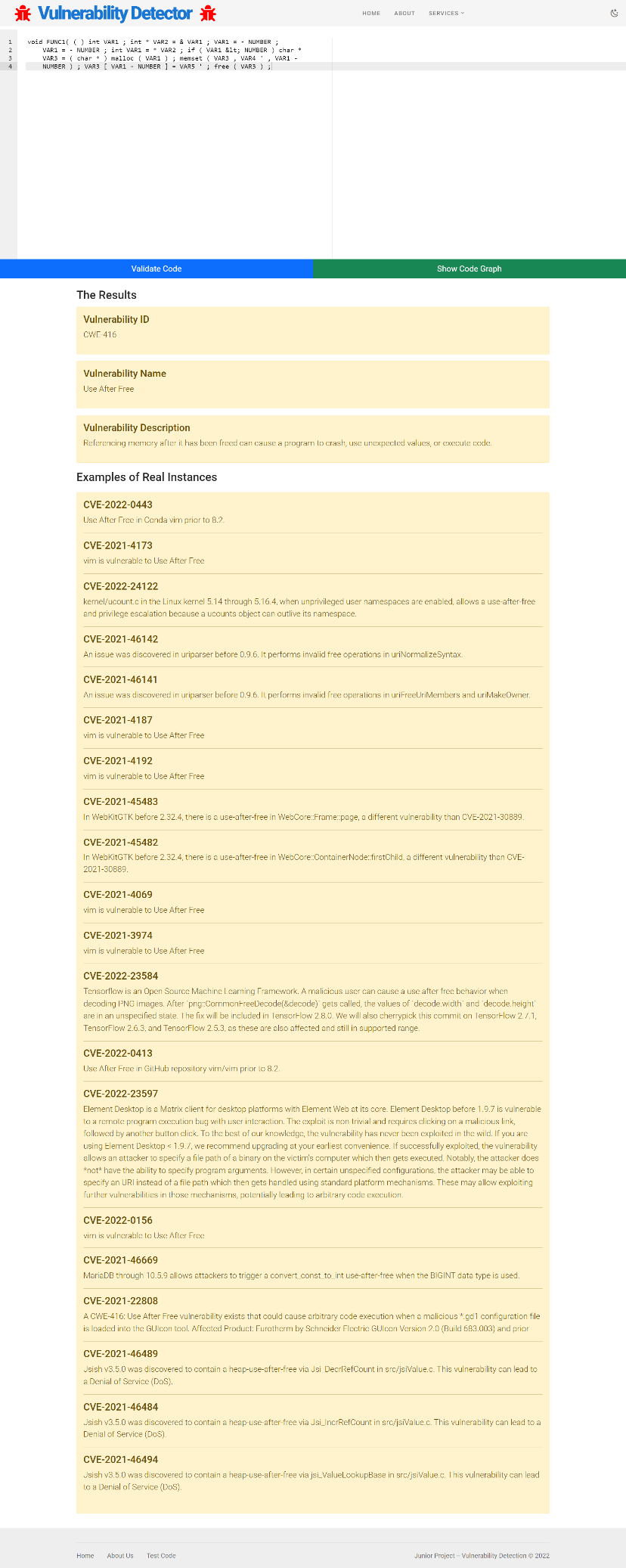
### 5.6.1 Home



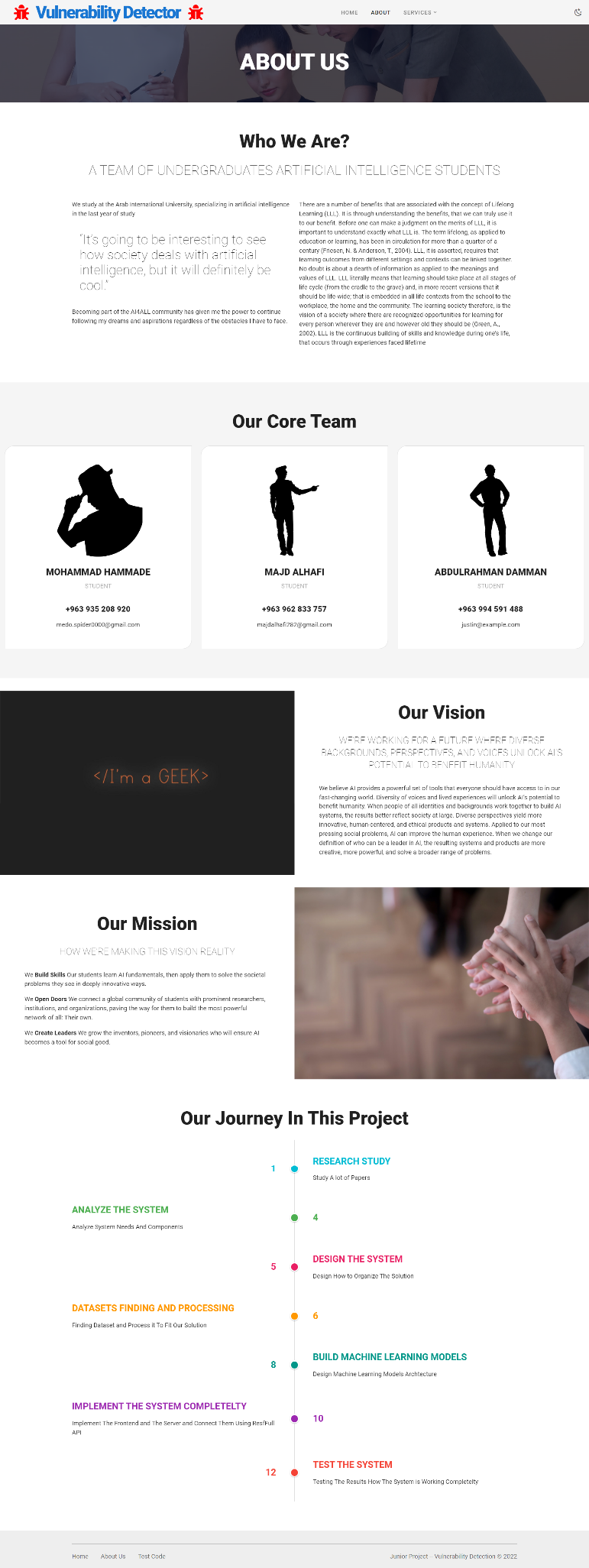
### 5.6.2 Github Repository



### 5.6.3 Code Editor & Results



### 5.6.4 About as

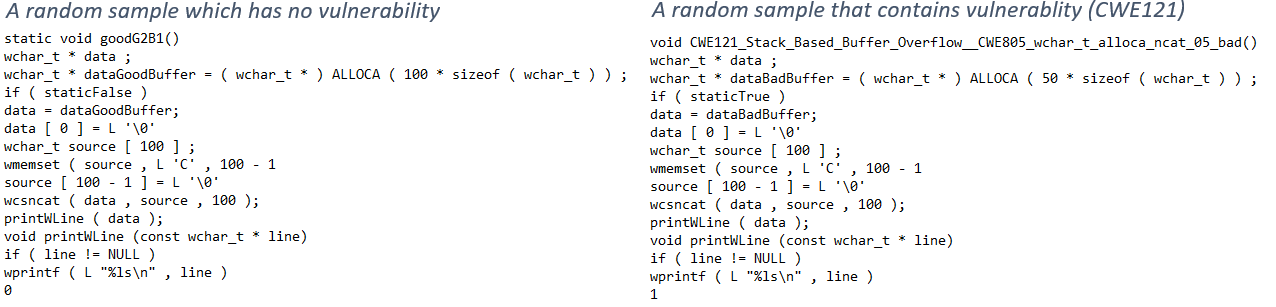


## 5.7 Dataset

In SySeVR [31] system researchers collected the Semantics-based Vulnerability Candidate (SeVC) dataset, which contains all kinds of vulnerabilities that are available from the National Vulnerability Database (NVD) and the Software Assurance Reference Dataset (SARD). At a high level, the SyVC representation corresponds to a piece of code in a program that may be vulnerable based on a syntax analysis. The SeVC representation corresponds to the extended statements of the SyVCs, with the extension to incorporate some of the other statements that are semantically related to the SyVCs. SeVC dataset focuses on 1,591 open source C/C++ programs from the NVD and 14,000 programs from the SARD. It contains 420,627 SeVCs, including 56,395 vulnerable SeVCs and 364,232 SeVCs that are not vulnerable. The dataset is stored in text files in this form. Where the last row indicates whether the sample is vulnerable "1" or clear "0". The type of the vulnerable was stored by its ID assigned by Common Weakness Enumeration (CWE) organization (e.g. CWE-78: Improper Neutralization of Special Elements used in an OS Command). [46]

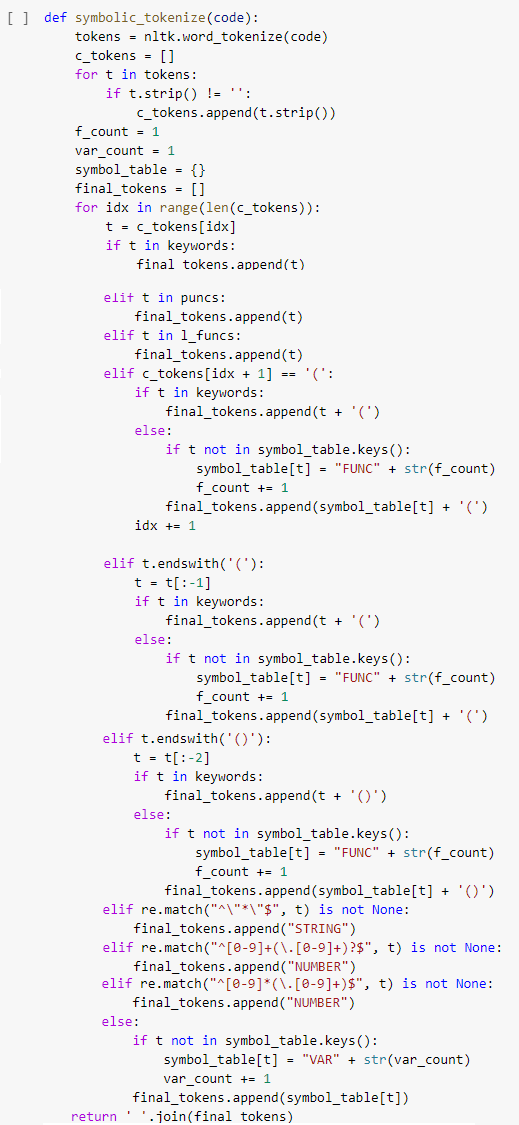
Description of CWE examples





Because a programming language is a context-free grammar, the variable namespace is open. This leads to an increase in the number of tokens, even if the tokens have the same role. Therefore, regular expressions were used in order to identify the names of the variables and functions to convert them into fixed tokens. For example, if the first source code has a variable "X" and the second source code has a variable "Y". In this case, the tokenizer will assign an ID for the token "X" different than the token "Y". But "X" and "Y" are variables and they do the same role in any programming language. If they are replaced by a constant word such as "VAR" for variables and “FUNC” for functions, so in all samples the first variable will be named VAR1 and the second will be named VAR2 and etc. then the tokenizer will give them the same ID for all variables in the source codes. In this case the models will learn the structure of code that contains vulnerable.

Function that process the source code



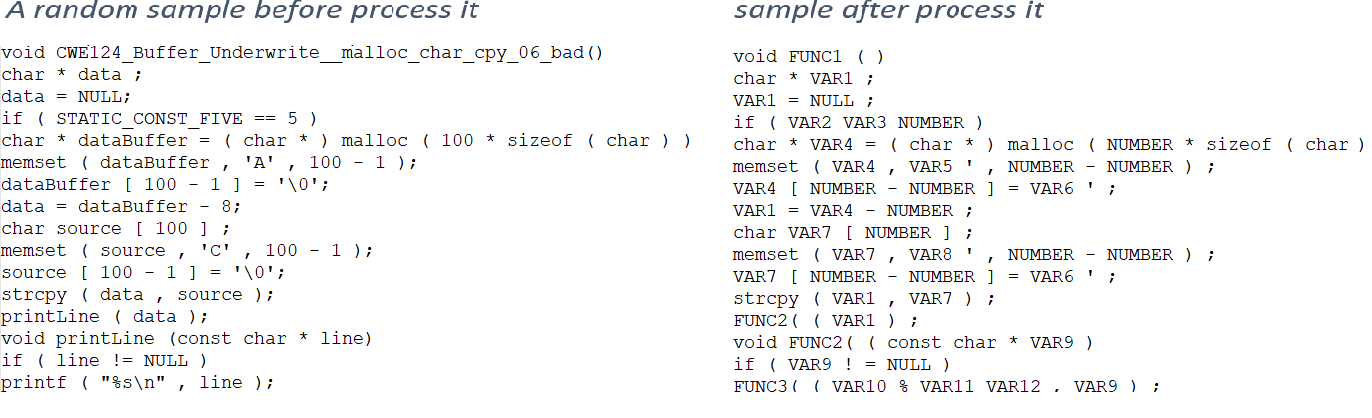


Table 2 Statistical information about dataset’s classes

|  |  |
| --- | --- |
| Classes | Number of samples |
| 0 | 88433 |
| 1 | 30616 |

Statistical information about top 10 vulnerable classes

|  |  |
| --- | --- |
| Class | Number of samples |
| CWE121 | 4930 |
| CWE122 | 4232 |
| CWE134 | 2477 |
| CWE124 | 1947 |
| CWE127 | 1721 |
| CWE194 | 1416 |
| CWE195 | 1270 |
| CWE78 | 1223 |
| CWE126 | 1082 |
| CWE690 | 905 |

As table 1 shown, there are approximately 88k samples that do not contain vulnerable. While on the vulnerable samples the class with the most samples has only 4930 samples (CWE121). So, using any augmentation technique will not make any sense. Because it's too hard to add about 83k samples to each other classes. The best solution to solve this problem is to divide the main task into two tasks, each task is done by a model, by fragmenting the dataset into two several parts, one contains whole samples and the model trained over it to detect if the source code has vulnerable or not (Binary classification), while the other fragment contains only vulnerable samples to detect the type of this vulnerable (Multiclass classification). But even after fragmenting, the two fragments suffer an imbalanced classes issue, this makes the models will have a high chance to overfit data. Synthetic Minority Oversampling Technique (SMOTE) was used to overcome that problem, SMOTE first selects a minority class instance at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b [47].

First fragment after applying SMOTE.

|  |  |
| --- | --- |
| Classes | Number of samples |
| 0 | 88433 |
| 1 | 88433 |

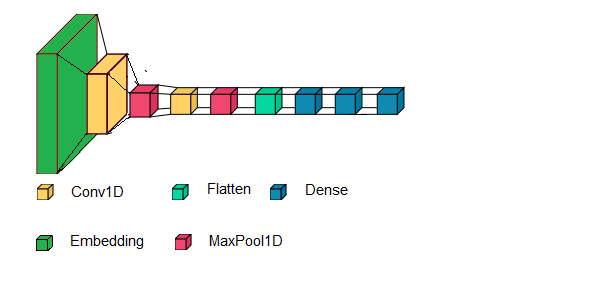
Second fragment applying SMOTE.

|  |  |
| --- | --- |
| Class | Number of samples |
| CWE121 | 4930 |
| CWE122 | 4930 |
| CWE134 | 4930 |
| CWE124 | 4930 |
| CWE127 | 4930 |
| CWE194 | 4930 |
| CWE195 | 4930 |
| CWE78 | 4930 |
| CWE126 | 4930 |
| CWE690 | 4930 |

## 5.8 Models

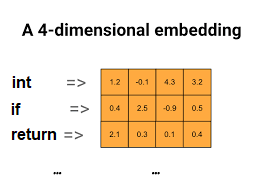
### 5.8.1 Binary classification model

The model consists of one embedding layer, two 1-D convolution layers, 3 fully connected layers. The input to the network is a preprocessed text, where each word embedded into 13-D vector, and the total number of words are truncated into 500 words. The number of layers was selected so as to maintain a high level of accuracy while still being fast enough for real-time purposes. In addition, it utilized max pooling, too.



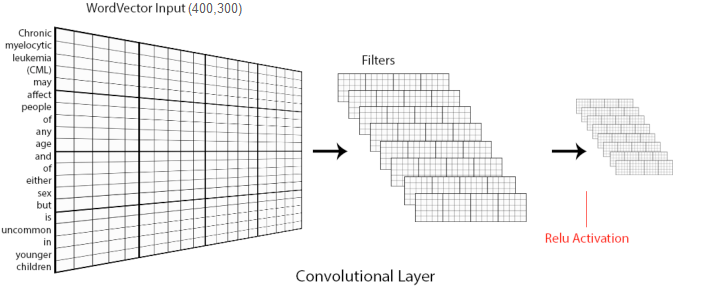
* Embedding layer

An embedding learns tries to find the optimal mapping of each of the unique words to a vector of real numbers. Where words that occur in the same context will have similar vector.



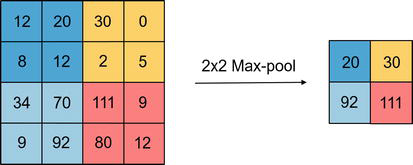
* Convolution layer

The main component of CNN is the convolution layer. This layer takes the vectors of words and applies filters on these vectors to extract local and position-invariant features.



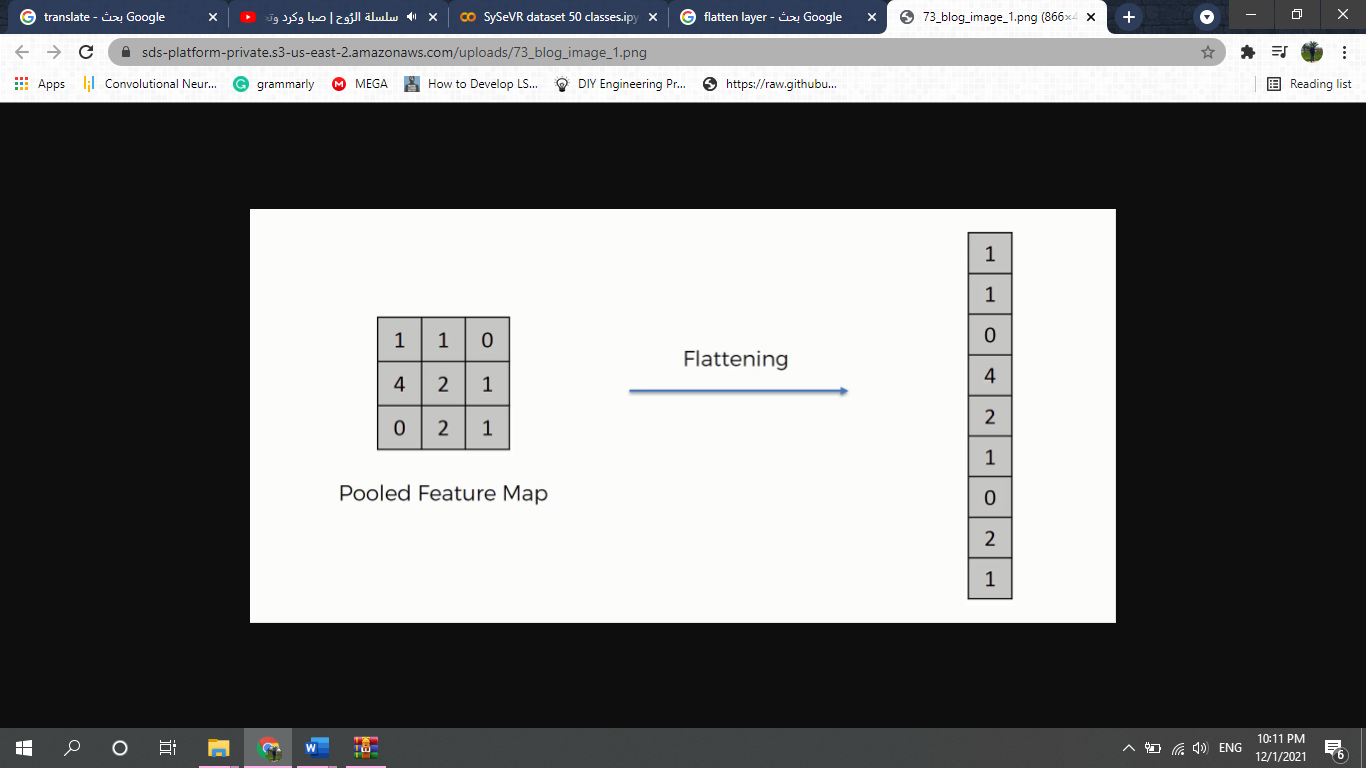
* Max pooling

Pooling mainly helps in extracting sharp and smooth features. It is also done to reduce variance and computations. And it is a part to help to reduce overfitting.



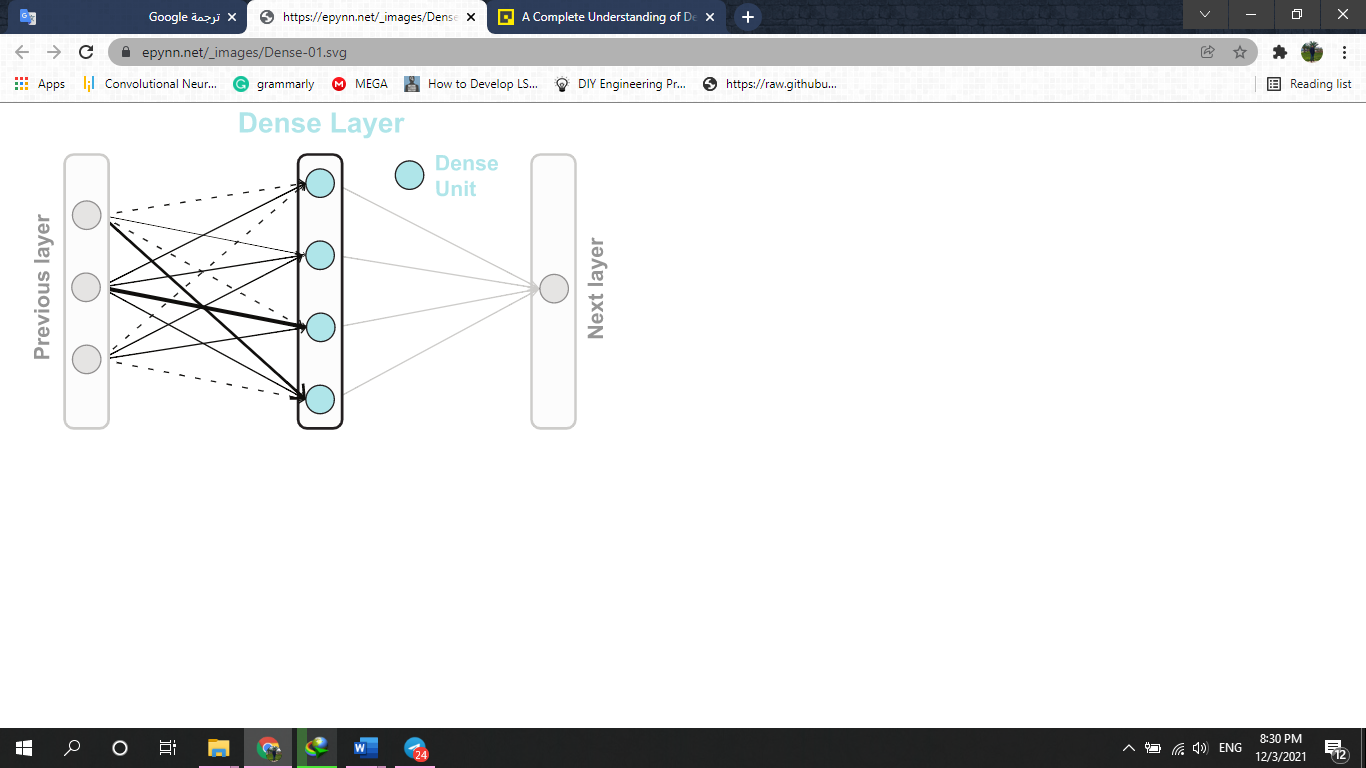
* Flatten layer

Flatten is used to flatten the input, such as converting the data into a 1-dimensional array as input to the next layer. It flattens the output of the convolutional layers to create a single long feature vector and it is connected to the final classification model layer, which is called a fully-connected layer.



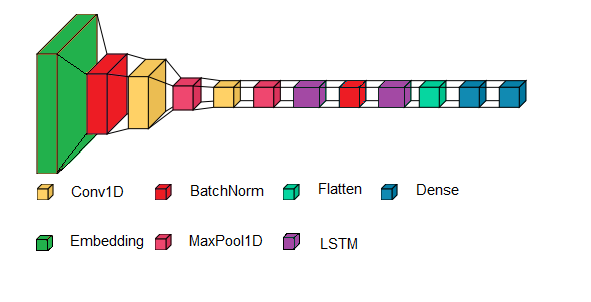
* Dense Layer

It is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. This layer is the most commonly used layer in artificial neural network.



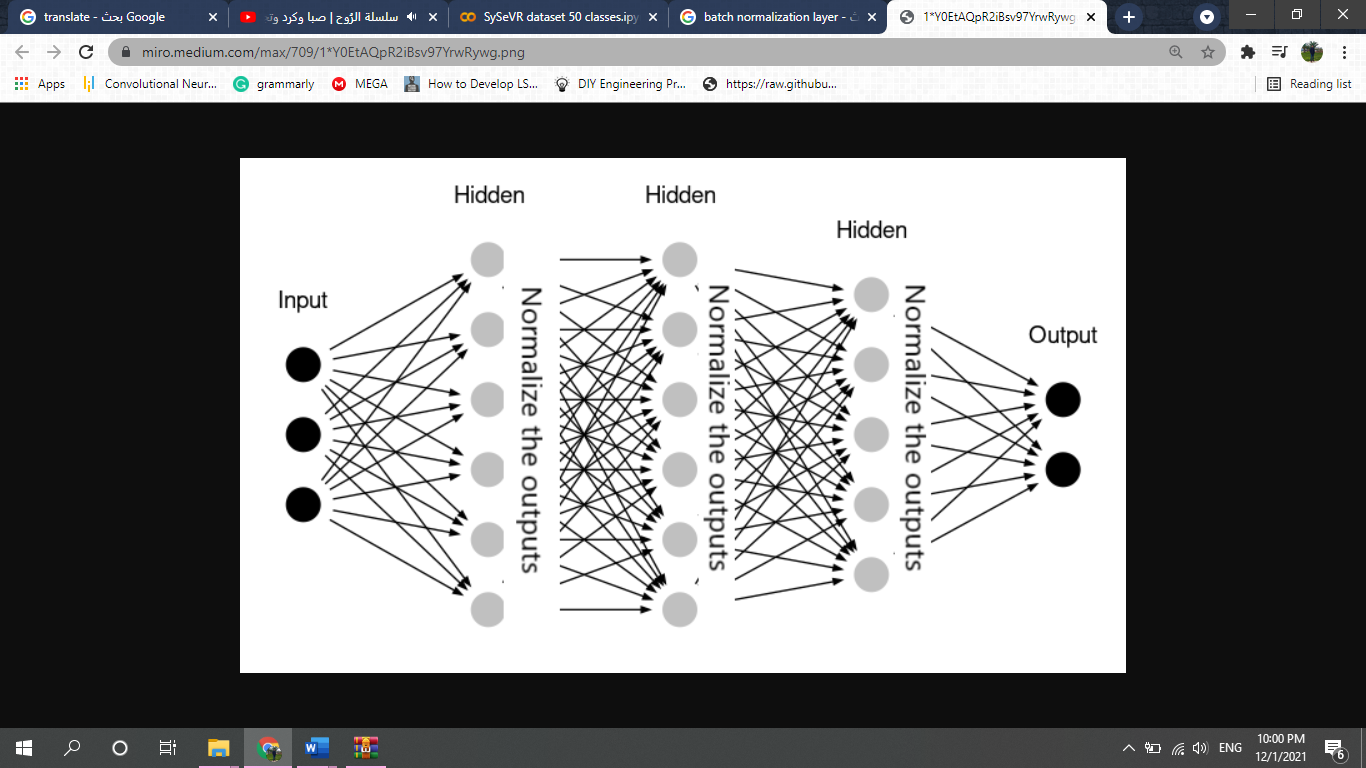
### 5.8.2 Multiclass classification model

The network consists of two 1-D convolution layers, two LSTM layers, and two fully connected layers. The input to the network is a preprocessed text, where each word embedded into 300-D vector, and the total number of words are truncated into 400 words. The number of layers was selected so as to maintain a high level of accuracy while still being fast enough for real-time purposes. In addition, it utilized max pooling and batch normalization more effectively in order to minimize overfitting.



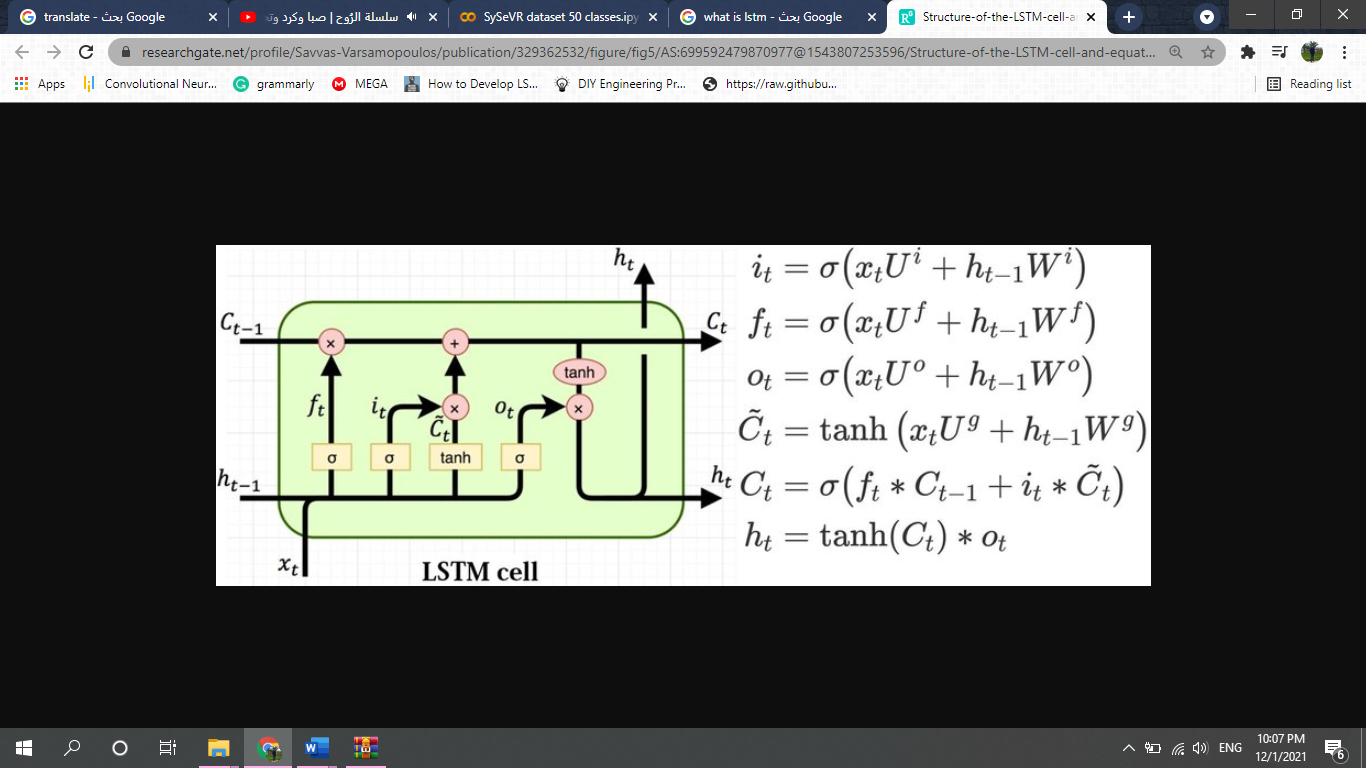
* Embedding layer
* Batch normalization layer

Batch normalization solves a major problem called internal covariate shift. It helps by making the data flow between intermediate layers of the neural network. It has a regularizing effect which means you can often remove dropout.



* Convolution layer
* Max pooling
* LSTM

LSTM networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. It is applied in complex problem domains like text classification, machine translation, speech recognition, and more.



* Flatten layer
* Dense layer

# Chapter 6: Result & Conclusion

## 6.1 Results

### 6.1.1 Binary Classification Model

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | Model | Accuracy | | Hyperparameters | | | | | | |
| Train | Test | Optimizer | Batch | Epoch | Loss function | | Learning Rate | Activation function |
| MSR [48] | DT | 98% | 84% | - | | | | | | |
| RF | 99% | 94% | - | | | | | | |
| Convolution + LSTM | 96% | 95% | Adam | 64 | 15 | | CrossEntropy | 0.001 | ReLU |
| VDISC [29] | RF | 99% | 93% | - | | | | | | |
| Chrome & Debian [49] | Convolution + LSTM | 93% | 93% | Adam | 32 | 22 | | CrossEntropy | 0.001 | ReLU |
| SySeVR [1] | Decision Tree | 99% | 97% | - | | | | | | |
| RF | 99% | 98% | - | | | | | | |
| LR | 76% | 76% | - | | | | | | |
| Convolution + LSTM | 97% | 97% | Adam | 32 | 50 | | CrossEntropy | 0.001 | ReLU |
| Convolution | 74% | 74% | RMSprop | 128 | 10 | | CrossEntropy | 0.005 | Tanh |
| 74% | 74% | Adam | 32 | 10 | | CrossEntropy | 0.001 | Tanh |
| 74% | 74% | Adagrad | 16 | 10 | | CrossEntropy | 0.0001 | ReLU |
| 99% | 98% | Adam | 64 | 10 | | CrossEntropy | 0.005 | ReLU |
| 99% | 99% | Adam | 64 | 100 | | CrossEntropy | 0.09 | ReLU |

As shown in the table, the best accuracy that was reached is 99% using 2 convolution layers where number of kernels are 256,128 and kernel size is 7, with 3 dense layers that contain 64,16,1 hidden neurons in each of them where the activation function is Tanh for the last layer to make the classification, trained on SySeVR dataset [1].

#### **6.1.1.1 Binary classification discussion**

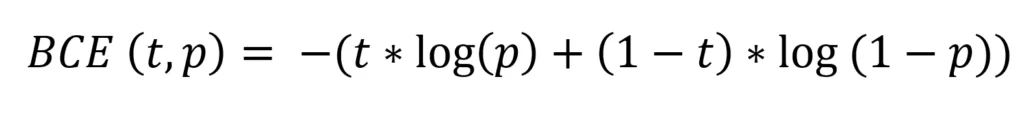
ReLU function has been relied on as an activation function so that it is considered the most widely used activation function in neural networks nowadays, and It is known that one of the greatest advantages of ReLU has over others is that it does not activate all neurons at the same time. Although other functions were tried, such as Tangent, but through experiments, it was found that it gives the best and the fastest results, ReLU converges six times faster than tanh activation functions.

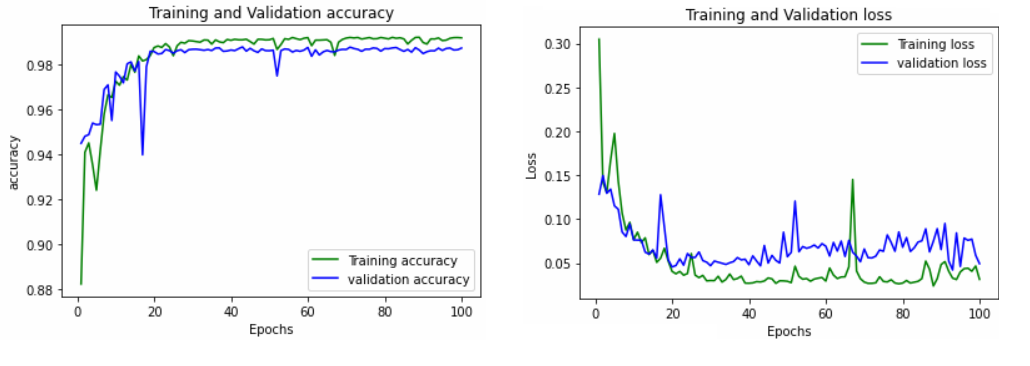
A traditional default value for the learning rate is 0.001, but experiments have shown that the best learning rate is 0.09.

Many experiments were tried, and articles have been read about the effect of the batch size, and it was found that the best results are when the batch equals 64.

Adam combines the best properties of the AdaGrad that improves performance on problems with sparse gradients and RMSProp that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). Adam optimizer can handle sparse gradients on noisy problems.

The cross entropy loss function is a measure of the difference between two probability distributions for a given random variable or set of events. There are two types of cross entropy one for binary label and the other for multiclass label. This study proposed cross entropy as a loss function.





The figure above shows the accuracy curve and the loss curve in both train set and test set during epochs

### 

### 6.1.2 Multiclass Classification Model

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | Model | Accuracy | | Hyperparameters | | | | | |
| Train | Test | Optimizer | Batch | Epoch | Loss function | Learning Rate | Activation function |
| MSR [48] | Convolution + LSTM | 74% | 48% | Adam | 32 | 100 | CrossEntropy | 0.001 | ReLU |
| SySeVR [1] | SVM | 43% | 41% | - | | | | | |
| Decision Tree | 95% | 95% | - | | | | | |
| Convolution + LSTM | 76% | 76% | Adagrad | 64 | 10 | CrossEntropy | 0.001 | Tanh |
| 87% | 86% | Adam | 128 | 20 | CrossEntropy | 0.001 | ReLU |
| 95% | 95% | RMSprop | 16 | 10 | CrossEntropy | 0.001 | ReLU |
| 97% | 96% | Adam | 32 | 50 | CrossEntropy | 0.0005 | ReLU |
| 98% | 98% | Adam | 32 | 50 | CrossEntropy | 0.001 | ReLU |

Also, SySeVR dataset [1] gave our solution the best accuracy 98% using 2 convolution where the number of kernels are 64,128 and the kernel size is 3, with 2 LSTMs of 100,10 hidden neurons to maintain the sequence characteristics, and 2 dense layers of 100,50 neurons at the last of model architecture where the activation function of it is sigmoid to compute the probability of each class among the sample.

#### **6.1.2.1 multiclass classification discussion**

ReLU function has been relied on as an activation function so that it is considered the most widely used activation function in neural networks nowadays, and It is known that one of the greatest advantages of ReLU has over others is that it does not activate all neurons at the same time. Although other functions were tried, such as Tangent, but through experiments, it was found that it gives the best and the fastest results, ReLU converges six times faster than tanh activation functions.

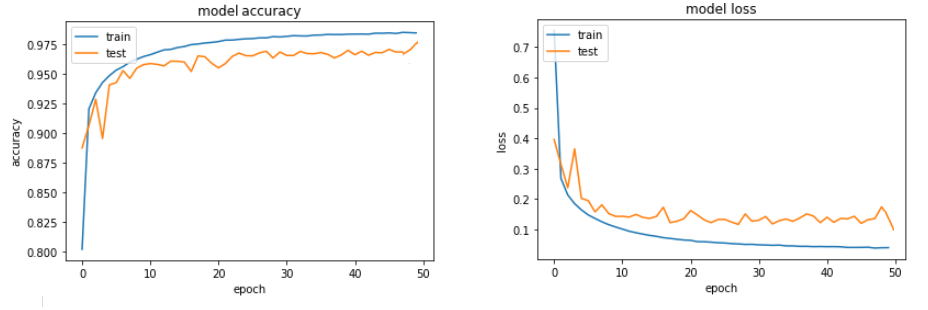
Several learning rate values were tried such as 0.0005,0.01 and 0.001, but the default value 0.001 is the best.

Many experiments were tried to figure the effect of batch size such as 128,64,32 and 16. It was found that the best results are when the batch size equals 32.

Adam combines the best properties of the AdaGrad that improves performance on problems with sparse gradients and RMSProp that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). Adam optimizer can handle sparse gradients on noisy problems [50].

The cross entropy loss function is a measure of the difference between two probability distributions for a given random variable or set of events. There are two types of cross entropy one for binary label and the other for multiclass label. This study proposed categorical entropy as a loss function.





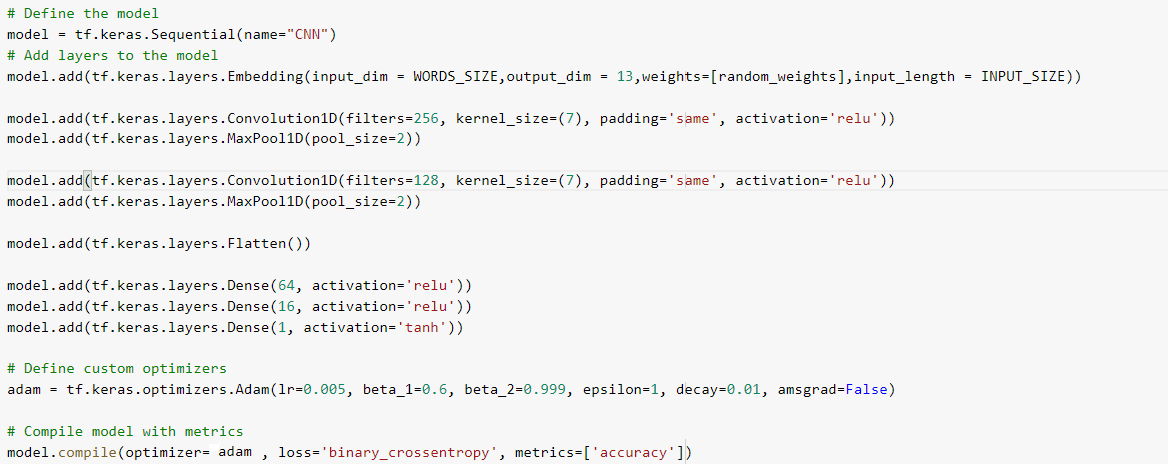
The figure above shows the accuracy curve and the loss curve in both cases train set and test set during epochs

## 6.2 Results Comparison

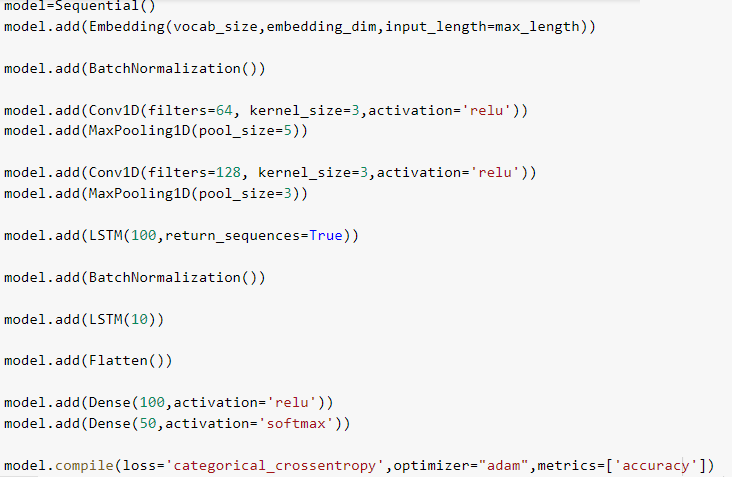
|  |  |  |  |
| --- | --- | --- | --- |
| Project | Dataset | Model | Accuracy |
| Deep Learning based Vulnerability Detection: Are We There Yet? [13] | ReVeal [28] | GGNN | 84% |
| Security Vulnerability Detection Using Deep Learning Natural Language Processing [36] | NVD/SARD [30] | BERT+BLSTM | 93% |
| SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities [31] | SySeVR [1] | BGRU | 96% |
| Automated Vulnerability Detection in Source Code Using Deep Representation Learning [35] | Draper VDISC [29] | CNN+RF | 91.6% |
| Combining Graph Neural Networks with Expert Knowledge for Smart Constract Vulnerablility Detection [33] | ESC and VSC [26] | GCE | 89% |
| Multi-context Attention Fusion Neural Network for Software Vulnerability Identification [32] | SARD [51] | Attention Fusion Model | 99% |
| This project | SySeVR [1] | CNN | 99.2% |
| Convolutional + LSTM | 98.7% |

## 6.3 Code snip

### 6.3.1 binary classification model



### 6.3.2 multiclass classification model



## 

# Conclusion and Future Works

The system was trained to detect the vulnerability and classify it into a class among 50 classes in source code written in C/C++. This study looks forward to adding new features to the system such as representing the code as graph and apply graph neural network to detect vulnerabilities, adding new types of vulnerabilities, detecting multiple vulnerabilities, and also adding other programming languages in addition to C/C++. In order to provide the best utilization and help to the user, it is not enough to only discover the vulnerability, but also to detect the location of the vulnerability in the source code. Then fixing the code using sequence to sequence model to produce a non-vulnerable code.

# References

|  |  |
| --- | --- |
| [1] | [Online]. Available: https://github.com/SySeVR/SySeVR. |
| [2] | I. V. Krsul, in *SOFTWARE VULNERABILITY ANALYSIS*, 1998, p. 1. |
| [3] | I. Chen and C. Mosiman, "facilitiesnet," 2019. [Online]. Available: https://www.facilitiesnet.com/buildingautomation/article/How-Hackers-Exploit-Cybersecurity-Vulnerabilities--18317. |
| [4] | T.-S. Chou, "SECURITY THREATS ON CLOUD COMPUTING VULNERABILITIES," *International Journal of Computer Science and Information Technology ( IJCSIT ),* vol. 5, 2013. |
| [5] | K. Chivers, "norton," 8 Aug 2019. [Online]. Available: https://us.norton.com/internetsecurity-emerging-threats-how-do-zero-day-vulnerabilities-work-30sectech.html. |
| [6] | " OWASP," [Online]. Available: https://owasp.org/www-community/vulnerabilities/Buffer\_Overflow. |
| [7] | [Online]. Available: https://portswigger.net/web-security/cross-site-scripting. |
| [8] | "portswigger," [Online]. Available: https://portswigger.net/web-security/sql-injection. |
| [9] | "ctf101," [Online]. Available: https://ctf101.org/binary-exploitation/what-is-a-format-string-vulnerability/. |
| [10] | "securecoding," [Online]. Available: https://www.securecoding.com/blog/integer-overflow-attack-and-prevention/. |
| [11] | "synopsys," [Online]. Available: https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/. |
| [12] | "snyk," [Online]. Available: https://snyk.io/learn/application-security/static-application-security-testing/. |
| [13] | S. Chakraborty, R. Krishna, Y. Ding and B. Ray, "Deep Learning based Vulnerability Detection: Are We There Yet?," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. TBD, 2020,* 2020. |
| [14] | O. Harris, "softwaresecured," 2019. [Online]. Available: https://www.softwaresecured.com/top-sast-tools-for-developers/. |
| [15] | S. Koussa, "techbeacon," [Online]. Available: https://techbeacon.com/app-dev-testing/13-tools-checking-security-risk-open-source-dependencies. |
| [16] | E. Katz, "spectralops," 2020. [Online]. Available: https://spectralops.io/blog/top-12-open-source-code-security-tools/. |
| [17] | ". B. D. A. S. T. (. Software", "Software Testing Help," 2021. [Online]. Available: https://www.softwaretestinghelp.com/dynamic-application-security-testing-dast-software/#1\_Netsparker\_Recommended\_Tool. |
| [18] | [Online]. Available: https://www.shiftleft.io/. |
| [19] | ". A. I. t. I. S. C. Vulnerabilities", "cyberstartupobservatory.," [Online]. Available: https://cyberstartupobservatory.com/leveraging-artificial-intelligence-to-identify-source-code-vulnerabilities/. |
| [20] | [Online]. Available: https://www.ptsecurity.com/ww-en/products/ai/. |
| [21] | [Online]. Available: https://snyk.io/product/snyk-code/. |
| [22] | P. E. Black, "A Software Assurance Reference Dataset:," *Journal of Research of National Institute of Standards and Technology ,* p. 1, 2018. |
| [23] | S. Zhang, D. Caragea and X. Ou, "SpringerLink," 2011. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-23088-2\_15. |
| [24] | "Github," [Online]. Available: https://github.com/secureIT-project/CVEfixes. |
| [25] | [Online]. Available: https://paperswithcode.com/dataset/manysstubs4j. |
| [26] | "ESC and VSC dataset," [Online]. Available: https://github.com/Messi-Q/GPSCVulDetector/tree/tkde-1/data. |
| [27] | [Online]. Available: https://github.com/vntchain/GNNSCVulDetector. |
| [28] | [Online]. Available: https://github.com/VulDetProject/ReVeal. |
| [29] | [Online]. Available: https://osf.io/d45bw/. |
| [30] | [Online]. Available: https://nvd.nist.gov/. |
| [31] | Z. Li, D. Zou, S. Xu, H. Jin, I. Fellow, Y. Zhu and Z. Che, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilitie," *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING ,* 2021. |
| [32] | A. Tanwar, H. Manikandan, K. Sundaresan, P. Ganesan, S. K. Chandrasekaran and S. Ravi, "Multi-context Attention Fusion Neural Network for Software Vulnerability Identification," *Cisco Systems,* 2021. |
| [33] | Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu and a. X. Wang, "Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection," *IEEE Transactions on Knowledge and Data Engineering,* 2021. |
| [34] | Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, Wang, Sujuan, Zhong and Z. D. a. Yuyi, "VulDeePecker: A Deep Learning-Based System for," *Network and Distributed Systems Security (NDSS) Symposium 2018,* 2018. |
| [35] | R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich1, J. A. Harer, O. Ozdemir, P. M. Ellingwood and M. W. McConley, "Automated Vulnerability Detection in Source Code Using Deep Representation Learnin," *17th IEEE International Conference on Machine Learning and Applications,* 2018. |
| [36] | N. Ziems and S. Wu, "Security Vulnerability Detection Using Deep Learning Natural Language Processing," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS),* 2021. |
| [37] | E. Mashhadi and H. Hemmati, "Applying CodeBERT for Automated Program Repair of Java Simple Bugs," *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR),* 2021. |
| [38] | S. Singh, "Analytics Vidhya," 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/. |
| [39] | H. M. Wallach, "Topic modeling: beyond bag-of-words," p. 977–984, 2006. |
| [40] | F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," pp. 61-80, 2008. |
| [41] | "https://www.techopedia.com/definition/22431/abstract-syntax-tree-ast," 13 June 2018. [Online]. Available: https://www.techopedia.com/definition/22431/abstract-syntax-tree-ast. |
| [42] | pp-pankaj, "Geeksforgeeks," 2019. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/. |
| [43] | "Shiftleft," [Online]. Available: https://docs.shiftleft.io/core-concepts/code-property-graph. |
| [44] | T. S. Jepsen, "towards data science," 2018. [Online]. Available: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780. |
| [45] | P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, "Graph Attention Networks," 2018. |
| [46] | "Common Weakness Enumeration," [Online]. Available: https://cwe.mitre.org/about/index.html. |
| [47] | J. Brownlee, in *Imbalanced Learning: Foundations, Algorithms, and Applications*, 2013, p. 47. |
| [48] | [Online]. Available: https://github.com/ZeoVan/MSR\_20\_Code\_vulnerability\_CSV\_Dataset. |
| [49] | [Online]. Available: https://drive.google.com/drive/folders/1KuIYgFcvWUXheDhT--cBALsfy1I4utOy. |
| [50] | J. Brownlee, "machine learning mastery," [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. |
| [51] | [Online]. Available: https://samate.nist.gov/SARD/. |

**ملخص**

يعد أمان التطبيقات جزءًا أساسيًا من تطوير البرامج الحديثة. مع زيادة تعقيد الإنترنت ، يتجه المهاجمون أكثر فأكثر إلى الثغرات الأمنية ونقاط الضعف المعروفة في البرامج نفسها. لتجنب انتهاكات البيانات ، تحتاج الشركات إلى بناء الأمان في جميع مراحل بناء برامجها واختبارها ونشرها. هناك تقنيات مختلفة لاكتشاف الثغرات الأمنية مثل اختبار أمان التطبيقات الثابتة واختبار أمان التطبيقات الديناميكي، لكن هذه الحلول تعاني من ارتفاع معدلات إيجابية كاذبة وسلبية كاذبة عالية. كان الباحثون مهتمين بتطوير نظام قائم على الذكاء

BLSTM وBERTالاصطناعي لاكتشاف نقاط الضعف باستخدام نماذج التعلم العميق مثل

في هذا المشروع ، تم تطوير نموذجين للتعلم العميق أحدهما لاكتشاف ما إذا كانت شفرة المصدر تحتوي على أي ثغرة أمنية (نموذج تصنيف ثنائي ) والآخر (نموذج تصنيف متعدد الفئات) لتصنيف هذه الثغرة الأمنية. يتكون LSTM ويتكون نموذج التصنيف متعدد الفئات من طبقات تلافيفية و .CNNنموذج التصنيف الثنائي من .SySeVR تم تدريب كلاهما على مجموعة بيانات

بالنسبة لنموذج التصنيف الثنائي ، تبلغ الدقة 98.63٪. بالنسبة لنموذج التصنيف متعدد الفئات ، تبلغ الدقة 98٪ لتصنيف 50 نوعًا مختلفًا من الثغرات الأمنية.

**الجامعة العربية الدولية**

الجامعة العربية الدولية AIU جامعة سورية خاصة أُحدثت عام 2005، خططها الدراسية والوثائق الصادرة عنها معتمدة ومصدقة من قبل وزارة التعليم العالي في الجمهورية العربية السورية.

تعمل الجامعة على تحقيق الأهداف الآتية:

* إعداد جيل متميز من الخريجين الجامعيين القادرين على تلبية الحاجات النوعية للمجتمع والنهوض به.
* الإسهام في البحوث العلمية النظرية والتطبيقية التي تخدم أغراض التنمية الوطنية، ويتم العمل على حث الأساتذة والعاملين الأكاديميين على البحث العلمي والمشاركة في المؤتمرات والندوات التي تنظم الأبحاث.
* تحقيق الشراكة مع الجامعات العربية والأجنبية المرموقة بهدف التطوير والتحديث المستمرين للعمل الأكاديمي والقيام ببحوث علمية مشتركة.
* استقطاب الكفاءات الأكاديمية والبحثية المتميزة عن طريق توفير البيئة المناسبة لعملها.

**الجامعة العربية الدولية** من الجامعات السورية الأولى التي جرى تأسيسها وافتتاحها، وقد تمكنت من اجتذاب الكفاءات التعليمية والبحثية والإدارية المتميزة، لإنشاء صرح متكامل من النواحي الأكاديمية والتنظيمية والإدارية. وتمكنت من تخريج كوادر من المبدعين والمتميزين من خلال توفير  بيئة تعليمية ترتكز إلى مقومات نوعية ومادية فريدة منها:

* الخطط الدراسية الحديثة والمتطورة المستندة إلى نظام الساعات المعتمدة.
* الأطر التعليمية المنتقاة بعناية كبيرة.
* المختبرات العلمية الحديثة، ومختبر للمكتبات الإلكترونية.
* المحفزات المادية والمعنوية للطلبة.
* تطبيق طرائق التدريس التفاعلي.
* التوجيه والإرشاد الأكاديمي والتربوي.
* مجموعة كبيرة من اتفاقيات التعاون العلمي مع جامعات محلية وإقليمية ودولية ذات سمعة مرموقة.
* اتفاقيات ومذكرات تفاهم متعددة مع العديد من مؤسسات المجتمع المدني.
* الحرم الجامعي اللائق والمزود بكافة المرافق العلمية والرياضية والترفيهية، والذي نشجعك على زيارته والتعرف على مزاياه.
* الأنشطة والأندية الطلابية بمختلف أنواعها: الرياضية والثقافية والعلمية والاجتماعية.

**في الجامعة العربية الدولية** سنوات الحياة الجامعية هي وقت للاستثمار في مستقبل الطالب. فالمعارف والخبرات التي يحصلها في قاعة المحاضرات والمختبرات ستساعده في تطوير ذاته، وستمنحه أسباب النجاح في التخصص الذي اختاره، والنشاط الطلابي الذي يمارسه سيساعده في توسيع أفقه، وفعاليات التدريب والأندية والرياضة ستمكنه من تطوير مواهبه، ولربما تساعده في اكتشاف مواهب جديدة.

ليستثمر وقته وذهنه وروحه في جامعتنا كي يجني فوائد عمله والوقت الذي كرسه في السنين القادمة. ونحن سوف نكون بجانب طلبتنا في كل خطوة على دربهم.



**الجامعة العربية الدولية**

**كلية الهندسة المعلوماتية والاتصالات**

**مشروع ما قبل التخرج**

**كشف الثغرات**

تم تقديمه إلى

قسم الهندسة المعلوماتية

تقديم

**محمد حمادة محمد مجد الحافي   
 عبد الرحمن ضمان**

بإشراف

**المهندسة خلود الجلاد  
  
شباط 2022**

1. <https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer>

   2 <https://hakiri.io/>

   3 <https://www.whitesourcesoftware.com/> [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. [↑](#footnote-ref-5)
6. [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)
8. [↑](#footnote-ref-8)